Introduction

By including the SocketTools FileTransfer control in a project, setting some properties and responding to events, you can quickly and easily develop an application that uploads and downloads files. Here's the background and the process.

The File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP) provide a reliable means for sending and receiving files based on well-known and widely used standards. The SocketTools File Transfer control provides an interface to file transfer services, allowing developers to easily upload and download files, as well as perform remote file management functions, without requiring general knowledge of network programming or how the specific application protocols work.

The SocketTools File Transfer control provides a single interface that supports FTP and HTTP using standard Uniform Resource Locators (URL). For most applications, this is the only control that will be needed to upload and download files. However, in some cases a program may require the advanced features of a specific SocketTools control, such as posting a query to a web server. In this situation, the File Transfer control can be seamlessly integrated with the other SocketTools controls to build a more complex solution that requires a greater degree of customization.

The control is implemented as a standard COM object and is designed to be used in visual development tools as well as various scripting environments. Any programming language which can host ActiveX controls or create instances of a COM object should be capable of using the File Transfer control, such as Visual Basic, Visual C++, Visual FoxPro and PowerBuilder. Server and client-side scripting is also supported using languages such as VBScript and JScript. The control is completely self-contained and does not require developers to redistribute the Microsoft Foundation Classes (MFC) or Visual C runtime libraries, nor any other third-party library.

Windows Sockets

The Windows Sockets specification was created by a group of companies, including Microsoft, in an effort to standardize the TCP/IP suite of protocols under Windows. 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 would continue to work.

There are two general approaches that you can take when creating a file transfer program. One is to code directly against the Windows Sockets API. The other is to use a high-level component which provides a simpler interface to the library by setting properties 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 network programming. By including the File Transfer control in a project, setting some properties and responding to events, you can quickly and easily develop an application that uploads and downloads files.

Transmission Control Protocol (TCP)

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.

The Transmission Control Protocol (TCP) offers a full-duplex byte stream which may be read and written to in a fashion similar to reading and writing a file. TCP is used for both FTP and HTTP as the underlying network protocol to provide a reliable means of exchanging data between the local system and the server.

Protocol Standards

The standards that form the foundation for sending and receiving files over the Internet and corporate intranets are defined in documents called RFCs (Request For Comments) which describe how the various protocols should be implemented. At this writing, not all of the rules for secure FTP over TLS have reached the status of RFCs, but have been derived in part from Internet Engineering Task Force (IETF) drafts. The following documents were used when implementing the File Transfer control:

RFC 959 documents the File Transfer Protocol (FTP), which is used for file transfer between a client and a server, and for remote management of files on a server. The Internet draft document "Securing FTP with TLS" describes a mechanism that can be used by FTP clients and servers to implement security and authentication using the TLS protocol defined by RFC 2246 and the extensions to the FTP protocol defined by RFC 2228.

RFC 1945 documents Version 1.0 of the HyperText Transfer Protocol (HTTP), and RFC 2616 documents Version 1.1 of the protocol. These standards govern the communication of client applications such as browsers with web servers. The File Transfer Control implements the GET and PUT commands of these standards.

Uniform Resource Locator (URL)

The Uniform Resource Locator or URL is a way to identify a specific resource, such as a file. It is commonly used with web browsers, but can also be used to identify files on FTP servers as well. The File Transfer control supports the use of both HTTP and FTP URLs to make it easy to specify the location of a file on a file server. The protocol that is used is determined by the URL scheme. For file transfers using FTP, a URL will have the following format:

```
ftp://[username : password] @] host [:port] / [path / ...] [filename]
```

If no user name and password are provided, then the client session will be authenticated as an anonymous user. If a path is specified as part of the URL, the file will be located in that directory on the server. It's important to keep in mind that the paths in an FTP URL are relative to the home directory of the user account and are not absolute paths starting at the root directory on the server.

For file transfers using HTTP, a URL will have the following format:

```
http://[username : password] @] host [:port] / [path / ...] [filename]
```

A username and password are only required if access to the resource is restricted. If no user credentials are provided, then the client will not provide any authentication information to the server.

The File Transfer control supports the use of both types of URLs as the remote file name for **GetFile** (download) and **PutFile** (upload) operations. For simple file transfer operations that involve only a single file, a single call to either of these methods using a URL is all that is required.

File Transfer Control

The File Transfer control has properties and methods which can be used for two general functions: the transfer of files between a local machine and a remote machine on which a file or web server is running, and the management of remote files on an FTP server. The interface is designed to be simple and intuitive, yet flexible enough to handle a wide variety of development needs. The developer may choose to manage details of a connection to a server and subsequent file transfers explicitly, or may simply supply a URL in the format that would be acceptable to a browser. Multiple file transfers using wildcards are also supported with the File Transfer Protocol. The file management functions that are available while connected to an FTP server include obtaining lists of files and directories, creating directories, changing directories, removing directories and renaming and deleting files.

Service Ports

By convention, most file servers listen on the following ports:

Protocol	**Standard**	**Secure**
File Transfer Protocol	21	990
Hypertext Transfer Protocol	80	443

If you leave the ServerPort property at its default value of zero, then the FileTransfer control will automatically select the port according to the value of the ServerType and Secure properties. However, you can override the standard port values by setting the ServerPort property. This enables your application to establish a connection with a server that is configured to use non-standard port numbers.

Note that secure connections are not supported by the Freeware version of the File Transfer control. If your application requires a secure connection, then you can purchase a SocketTools development license that will provide you with that functionality.

Authentication

FTP servers generally require a user name and password in order to transfer or manage files during a session. Many servers support anonymous logins, in which a user name and password must still be supplied, but are not authenticated. If no user name and password are provided when establishing a connection, the control will use anonymous authentication by default.

HTTP servers generally do not require a user name and password in order to download files, but may require authentication when uploading files. By default, the File Transfer control encodes the user name and password for HTTP according to the Basic Authentication scheme.

Proxy Servers

In certain environments, connection to an FTP or HTTP server must be made explicitly through a proxy server. The File Transfer control supports this with a set of properties (**ProxyType**, **ProxyServer**, **ProxyPort**, **ProxyUser**, **ProxyPassword**) that must be set before attempting a connection. If the **ProxyType** is nonzero, then the remaining proxy-related properties are used by the control.

There is greater variation among proxies used for FTP than is the case for HTTP. If an FTP proxy is encountered that does not fall into one of the categories explicitly supported by the control, then the **fileProxyOther** proxy type should be used, and the Command method should be used to send any custom commands that are required to authenticate the user in accordance with instructions provided by the proxy vendor.

Uploading Files

The File Transfer Protocol supports uploading files from the client to the server in a universally implemented manner. The only limitations are those imposed by the server administrator and the access rights of the authenticated user.

Uploading files using the Hypertext Transfer Protocol can be more complicated because there are several different methods that are used and not all servers are configured to support file uploads. File uploads are typically implemented using either the PUT or POST command. Most web servers do not support the use of the PUT command by default. Those servers that do support the command usually require that the client be authenticated prior to permitting the transfer. The control's **PutFile** method uses this command.

The POST command is a more common method of uploading a file and it is what's used when a form displays a button that allows the user to browse for a file on the local system and then submit it to the web server for processing. The control's **PostFile** method uses this command. Note that the File Transfer control does not provide extensive support for submitting form data. For advanced functionality such as creating a virtual form and submitting the data to the server, it is recommended that you use the SocketTools component instead.

Persistence

When using the File Transfer Protocol, any number of file transfers may be performed during a single session. There are actually two connections established during a session. The first is the command

channel which is used to authenticate the session and issue commands. The second is the data channel, which is used during the actual data transfer during a file upload or download.

The Hypertext Transfer Protocol may use either transient or persistent connections. The **KeepAlive** property enables the application to indicate to the server whether or not the connection should persist. However, it is important to note that the server may choose to close the connection even if the client requests a persistent connection. If this occurs, the control will automatically attempt to reconnect to the server, but the application must be prepared to handle any potential errors when requesting multiple files from a web server.

Firewalls

Typically, firewalls are configured to allow connections to be made on the standard ports. HTTP business is transacted on a single connection, and rarely encounters firewall problems.

However, in accordance with the FTP standards, each FTP file transfer or file listing is transacted on a separate connection, using higher-numbered ports that are assigned dynamically from a pool of currently-used ports. These ports are assigned by the underlying network software. Independent of the network software, a firewall may be configured to block access to higher-numbered ports. The symptom of firewall blockage with FTP is that the client can login to the server, and can accomplish certain management functions, but no file transfers or file listings can be done.

If the firewall is on the server side, then the Boolean property **Passive** should be False. If the firewall is on the client side, then **Passive** should be True. If there are firewalls on both sides of the connection, then some relaxation in the firewall constraints must be implemented. For example, the firewall on the server side may be configured to allow inbound connections from the client on a limited number of ports.

The File Transfer control uses this basic model for file transfer:

Connect to the server using the **Connect** method.
Upload and/or download one or more files using **PutFile** or **GetFile**.
Disconnect from the server using the **Disconnect** method.

The File Transfer control has implemented this model both explicitly in multiple steps, as well as implicitly when using a URL. In addition, wild-carded multiple file transfers are supported for FTP. A developer may choose whichever approach best suits his application.

Redirection of download requests for HTTP is supported silently, provided that the server supplies sufficient information to support the redirection.

A connection to a server is established using the Connect method:

```
FileTransfer1.Connect [ServerName] [,ServerPort] [,UserName] [,Password]
[,Timeout] [,Options]
```

All of the parameters of the **Connect** method are optional. The values of missing parameters are taken from the current values of the properties of the same name.

The behavior of the **Connect** method can be further modified by other property values, namely:

The **Secure** property determines whether or not to negotiate a SSL connection.

The properties **ProxyServer**, **ProxyPort**, **ProxyUser**, and **ProxyPassword** are used to establish a connection through a proxy server, according to the value of **ProxyType**.

The type of server (FTP or HTTP) is inferred either from the **ServerType** property, or from the **ServerPort** property or parameter.

The file transfer is typically implemented using either the **PutFile** or **GetFile** methods:

```
FileTransfer1.PutFile LocalFile, RemoteFile
FileTransfer1.GetFile LocalFile, RemoteFile
```

These methods may be used repeatedly following a single connection to a server. Note that in the case of an HTTP server, the **KeepAlive** property should be set to a value of true in order to transfer multiple files on a single connection. Otherwise, the connection will be terminated at the completion of a single file transfer, and the client must create a new connection for a subsequent transfer.

The connection to the server is terminated by calling the Disconnect method:

```
FileTransfer1.Disconnect
```

Note that if a remote file is specified by a URL, then the connection need not be made explicitly. Instead, the control will silently connect before each file transfer.

When connected to an FTP server, a client application using the File Transfer control may transfer multiple files in a single operation.

Put- and/or get-of-multiple-files is implemented by:

```
FileTransfer1.PutMultipleFiles LocalDirectory, FileMask
FileTransfer1.GetMultipleFiles LocalDirectory, FileMask
```

When downloading files from an HTTP server, the FileTransfer control is able to apply certain forms of redirection information provided by the server, if the requested file has been moved. Namely, if the server signals the redirection by a 300-level response code that the HTTP standards have reserved for the purpose, and if the server also supplies a Location response header for the new location of the file, then the control will silently apply the redirection information to obtain the file.

Some servers may supply redirection information in the form of "meta tags" in an HTML document. The current version of the File Transfer control does not recognize and analyze meta tags.

By default, the File Transfer control overwrites existing files on the target machine. However, for an FTP transfer, the transferred data will be appended to the target file or files, provided that:

```
FileTransfer1.AppendFile = True
```

Some FTP servers support the ability to resume an interrupted transfer. In order to enable this capability from the File Transfer control, invoke **GetFile** or **PutFile** with an additional, optional argument, which is a byte offset:

```
FileTransfer1.PutFile LocalFile, RemoteFile, Offset
FileTransfer1.GetFile LocalFile, RemoteFile, Offset
```

Starting at the specified *Offset* within the source file, transferred data will be appended to the target file. This feature should be used only for binary file transfers. For ASCII file transfers, this feature should not be used, due to the potential transformation of line termination characters (between Windows and UNIX systems, for example).

The File Transfer control can be used to generate file listings when connected to an FTP server. The general model for generating file listings is:

OpenDirectory
ReadDirectory (in a loop)
CloseDirectory

The first parameter of the **OpenDirectory** method is required to be present. If it is an empty string ("" in Visual Basic), it will be interpreted as the current directory. In general, it may be a path name. It may specify a subset of files in a directory by use of a file mask.

Examples:

```
' Generate a listing for the current directory
FileTransfer1.OpenDirectory

' Generate a listing for a specified directory
FileTransfer1.OpenDirectory "/usr/myDirectory"

' Generate a listing for files in the current directory
FileTransfer1.OpenDirectory "*.txt"
```

Some FTP servers do not properly implement the generation of a file listing for a specified path, but can only deal with the current directory. If **ReadDirectory** returns improper results, try setting the current directory to the desired directory before **OpenDirectory**:

```
FileTransfer1.ChangeDirectory "/usr/myDirectory"
FileTransfer1.OpenDirectory
```

The **ReadDirectory** method has several output parameters:

```
ReadDirectory(FileName [,FileLength] [,FileDate] [,FileOwner] [,FileGroup]
[,FilePerms] [,IsDirectory])
```

The default behavior of the **ReadDirectory** method is to automatically detect the format of the file listing provided by the FTP server, and to parse each entry in the listing to fill its output parameters. The **DirectoryFormat** property may also be set explicitly to one of the supported formats before calling **OpenDirectory**, to force the control to interpret the listing provided by the server in a specified fashion. Note that this is rarely needed.

It is not required that you specify all of the arguments to the ReadDirectory method. Only the first output parameter of the **ReadDirectory** method is required:

```
' List only names and dates
FileTransfer1.ReadDirectory FileName,,FileDate
```

There are circumstances in which a developer may choose to interpret the file listing in the application code, rather than relying upon the control to parse the listing provided by the FTP server. These circumstances include:

The server deviates from the format that the control has auto-detected, or which the application has specified. In this case, entries that cannot be understood by the control will be skipped.

The server uses a directory format that is not supported by the control. In this case, entries that cannot be understood in any detail by the control will be skipped.

A directory contains an extremely large number of files. In this case, the control may exhaust memory that is dynamically allocated for retaining parsed directory entries before it has completed processing the directory.

In any of these circumstances, the application may specify the optional second parameter *ParseList* of the **OpenDirectory** method to be False, meaning "unparsed". When this is done, the unprocessed directory entries will be returned in the *FileName* output parameter, and the remaining output parameters may be omitted. Note that parameters that are not omitted will return empty strings.

If an application does not interrupt the **ReadDirectory** loop, and all entries are successfully processed, then **ReadDirectory** will eventually return an error indicating that the end of the directory listing has been reached.

If the **ParseList** property was set to a value of true, the **ReadDirectory** method will return a value of **fileErrorEndOfDirectory** when the last file has been returned. If the **ParseList** property is set to false, the method will return a value of **fileErrorEndOfData**.

Regardless of how the end of the listing data is indicated, the directory listing must be explicitly terminated by calling the **CloseDirectory** method.

```
nError = FileTransfer1.OpenDirectory(strDirName)
If nError > 0 Then
    MsgBox FileTransfer1.LastErrorStrong, vbExclamation
    Exit Sub
End If

Do
    nError = FileTransfer1.ReadDirectory(strFileName, _
        dwFileLength, strFileDate, _
        strFileOwner, strFileGroup, _
        dwFilePerms, bIsDirectory)

    If nError > 0 Then
        If nError <> fileErrorEndOfDirectory And _
           nError <> fileErrorEndOfData Then
           MsgBox FileTransfer1.LastErrorStrong, vbExclamation
        End If
        Exit Do
    End If
Loop

FileTransfer1.CloseDirectory
```

To get information about a single file, it is not necessary to list all of the files in a directory. The **GetFileStatus** method can be used instead to obtain information about a specific file. The first parameter specifies the name of the file and the remaining arguments are passed by reference and will contain information about the file when the method returns.

```
Dim strFileName As String
Dim nFileLength As Long
Dim strFileDate As String
Dim strFileOwner As String
Dim strFileGroup As String
Dim nFilePerms As Long
Dim bIsDirectory As Boolean
Dim nError As Long

nError = FileTransfer1.GetFileStatus( _
    strFileName, _
    nFileLength, _
    strFileDate, _
    strFileOwner, _
    strFileGroup, _
    nFilePerms, _
    bIsDirectory)
```

The File Transfer control can be used to manage remote files when connected to an FTP server. File name syntax must be that of the remote system.

Change Directory

The **ChangeDirectory** method changes the current working directory on the server.

```
FileTransfer1.ChangeDirectory "/user/myDirectory"
```

Create Directory

The **MakeDirectory** method creates a new directory on the remote FTP host.

```
FileTransfer1.MakeDirectory "/user/myNewDirectory"
```

Remove Directory

The **RemoveDirectory** method removes a directory on the remote FTP server.

```
FileTransfer1.RemoveDirectory "/user/myUnwantedDirectory"
```

Delete File

The **DeleteFile** method deletes an existing file from the remote FTP server.

```
FileTransfer1.DeleteFile "myFile"
```

Rename File

The **RenameFile** method changes the name of an existing file on the FTP server.

```
FileTransfer1.RenameFile "OldName", "NewName"
```

Quick Start Guide

This section is provided as a means to quickly get started with the File Transfer control. The examples provided in this section presume some familiarity with the Visual Basic programming language. However, the basic concepts are the same regardless of what language is used. Please refer to the technical reference for complete information on all of the properties, methods and constants used by the control. Before performing any of the steps in this guide, you should have installed the File Transfer control on your development system.

To include the control in your project in Visual Basic, simply select the **Project|Components...|Controls** menu option and select the SocketTools File Transfer. In other languages, follow the normal steps that are taken to include an ActiveX control in your development project.

In order to transfer files to or from a server, or to perform remote file management on a server, it is necessary to establish a connection to the server. The File Transfer control can establish the connection explicitly, with the **Connect** method. A connection can also be made implicitly using information inferred from a URL. The discussion in this section will be about explicit connections.

Regardless of how the connection is established, certain information is required. Information that is always required includes:

Server name
Server type
Server port
Whether or not the server is secure

The server name may be a string such as "ftp.sockettools.com" or "www.amazon.com". The string may also be an IP address, such as "128.121.218.65".

The server type is either FTP (File Transfer Protocol) or HTTP (HyperText Transfer Protocol).

The server port is a number associated with the service that is being provided. Usually, FTP and HTTP servers use port numbers that have been set by standards.

A secure server is one that supports the SSL (Secure Sockets Layer) protocol, or its successor, the TLS (Transport Layer Security) protocol. These protocols allow the parties to a network transaction to determine whether the other party will be "trusted", and to agree upon an encryption scheme that will be applied to all data exchanged during the transaction.

The server port, server type, and whether the server is secure are inter-related, and it is not generally necessary to explicitly specify all three. Here are the guidelines that describe the relationship among them:

If the server type is undefined, then the port number will be used to determine the server type. Namely, a port number of 21 or 990 corresponds to FTP, and a port number of 80 or 443 corresponds to HTTP. Any other port number will be treated as an error if the server type is not specified.

If the server port is specified as 0, then the server type and whether a secure connection is desired will determine the actual port number to be used. Namely, for non-secure FTP, port 21 will be used; for

secure FTP, port 990 will be used; for non-secure HTTP, port 80 will be used; for secure HTTP, port 443 will be used.

If a server port other than 21, 990, 80, or 443 is specified, then a server type must be explicitly specified. Otherwise, an error will result. In addition, the connection will be non-secure or secure according to whether security is explicitly specified.

Beyond the server name, server type, server port, and security, there are additional settings that may be relevant to establishing a connection.

FTP connections almost always require a user name and password. If neither is specified, the control will attempt an "anonymous" login. Not all FTP servers support anonymous login. In rarer circumstances, an account name may also be required. HTTP rarely requires a user name and password for downloads.

Either FTP or HTTP servers may be accessible only through a proxy server in certain circumstances. In those circumstances, a set of properties related to proxies must be specified before connecting. Please see the Technical Reference for further details.

Explicit connection to a server using the **Connect** method should be used if you plan to do multiple file transfers in a single session, or you wish to generate file listings or do other remote file management activities with an FTP server.

The **Connect** method has several parameters, all of which are optional:

```
Connect([ServerName] [,ServerPort] [,UserName] [,Password] [,Timeout]
[,Options])
```

If a given parameter is missing, then the current value of the corresponding property will be used in establishing the connection.

For example:

```
lResult = FileTransfer1.Connect(strServerName, nServerPort, _
                                strUserName, strPassword, _
                                nTimeout, lOptions)
```

And this is the same as this:

```
With FileTransfer1
    .ServerName = strServerName
    .ServerPort = nServerPort
    .UserName = strUserName
    .Password = strPassword
    .Timeout = nTimeout
    .Options = lOptions
    lResult = .Connect
End With
```

Note that there are properties that may affect establishing a connection, but are not available as parameters of the **Connect** method. Namely **ServerType**, **KeepAlive, Account** (for FTP), and the **Proxy**-related properties.

In the following examples, properties not explicitly mentioned are assumed to have their default values.

```
' Connect to a non-secure FTP server using a user-specified timeout
FileTransfer1.ServerType = fileServerFtp

lResult = FileTransfer1.Connect(editServerName.Text, , _
                                editUserNameText, editPasswordText, _
                                editTimeout.Text, lOptions)

If lResult <> 0 Then
    MsgBox "Connection attempt failed"  & vbCrLf & _
          FileTransfer1.LastErrorString, vbExclamation
    Exit Sub
End If
```

The **GetFile** method may be used after connecting to an FTP or HTTP server with the **Connect** method. The **GetMultipleFiles** method may only be used with an FTP server.

The code for the "Download" button reads the edit controls, connects to a server (FTP or HTTP), downloads a file, and disconnects:

```
Private Sub cmdDownload_Click()
    Dim lResult as Long

    On Error GoTo Err_Report
    FileTransfer1.UserName = editUserName.Text
    FileTransfer1.Password = editPassword.Text
    FileTransfer1.ServerPort = editPort.Text

    lResult = FileTransfer1.Connect(editServer.Text)
    If lResult <> 0 Then
        MsgBox "Connect Failed" & vbCrLf & FileTransfer1.LastErrorString
        Exit Sub
    End If

    lResult = FileTransfer1.GetFile(editLocalFile.Text, editRemoteFile.Text)

    If lResult <> 0 Then
        MsgBox "Download Failed" & vbCrLf & FileTransfer1.LastErrorString
    Else
        MsgBox FileTransfer1.TransferBytes & " bytes transferred"

    End If

    FileTransfer1.Disconnect
    Exit Sub

Err_Report:
    MsgBox Err.Number & ": " & Err.Description
    FileTransfer1.Disconnect
End Sub
```

The **PutFile** method may be used after connecting to an FTP or HTTP server with the **Connect** method. The **PutMultipleFiles** method may only be used with an FTP server.

SocketTools

The SocketTools File Transfer shares functionality with another Catalyst product called SocketTools. In addition to the file transfer and management functionality that the File Transfer control provides, SocketTools includes .NET assemblies, ActiveX controls and standard Windows libraries for many other popular Internet application protocols. There are several different editions of SocketTools available, and all editions provide royalty-free redistribution licensing and a thirty day money-back guarantee. Free evaluation copies can be downloaded from the Catalyst Development website at sockettools.com.

SocketTools .NET Edition

The SocketTools .NET Edition consists of managed code assemblies for use with .NET programming languages such as Visual Basic .NET, Visual C# and Delphi Prism. The product includes twenty classes which provide interfaces for various Internet protocols such as the File Transfer Protocol, Hypertext Transfer Protocol, Internet Message Access Protocol and Simple Mail Transfer Protocol. Using the .NET Edition you can easily transfer files, send and retrieve email messages, execute commands on servers and perform many other common tasks over the Internet. The SocketTools .NET classes are designed to be extremely simple to use without compromising performance, and are flexible enough to perform very complex tasks.

SocketTools ActiveX Edition

The SocketTools ActiveX Edition consists of ActiveX controls for use with visual development languages such as Visual Basic, Visual C++ and Delphi. A total of twenty controls provide client interfaces for the major application protocols such as the File Transfer Protocol, Simple Mail Transfer Protocol, Domain Name Service and Telnet. Visual Basic 6.0 is fully supported and the components can be used with any development tool that supports COM and the ActiveX control specification. The network controls support both synchronous (blocking) and asynchronous modes of operation, as well as advanced trace debugging facilities. All of the controls are thread-safe and can be used in multithreaded containers, such as Internet Explorer.

SocketTools Library Edition

The SocketTools Library Edition consists of standard dynamic link libraries, and can be used by virtually any Windows programming language that can call functions exported from a Windows DLL. A total of twenty libraries provide client interfaces for application protocols such as the File Transfer Protocol, Simple Mail Transfer Protocol and Telnet protocol. The API for the Library Edition is implemented with a simple elegance that makes it easy to use with any language, and is not just for C or C++ programmers. All of the libraries are thread-safe and can be used in multithreaded applications.