

FTP Client ActiveX Control

**File Transfer Protocol
(RFC 959)
ActiveX Control
for Microsoft® Windows™**

Version 5.2

**Copyright © 1995 - 2003 by Distinct Corporation
All rights reserved**

Distinct Corporation

3315 Almaden Expressway
San Jose, CA 95118 USA

Phone: +1 408-445-3270

Fax: +1 408-445-3274

Email: sales@distinct.com

WWW: <http://www.distinct.com>

Disclaimer

Distinct Corporation makes no warranties as to the contents of this documentation and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Information in this manual is subject to change without notice and does not represent a commitment on the part of Distinct Corporation. The Software described in this manual is furnished under a license agreement and may be used or copied only in accordance with the terms of that agreement.

Copyright Notice

© 1995 - 2003 by Distinct Corporation. All rights reserved.

No part of this publication may be reproduced, transmitted or translated into any language by any means without the express written permission of Distinct Corporation.

Trademarks

Distinct is a registered trademark and Visual Internet Toolkit and Distinct FTP Client are trademarks of Distinct Corporation. Windows is a registered trademark of Microsoft Corporation. Other product names are trademarks or registered trademarks of their respective owners.

Last updated June 5th, 2003

Published in the United States of America

Table of Contents

Table of Contents.....	3
1 FTP Client ActiveX Overview	7
1.1 Introduction	7
1.2 Usage of the FTP Client ActiveX Component	7
1.3 FTP Client Property Summary	10
1.4 FTP Client Event Summary.....	12
1.5 FTP Client Method Summary	13
1.6 D_FTP.TXT	14
2 FTP Client Properties	19
2.1 Account	19
2.2 Action	20
2.3 BinaryData	22
2.4 BlockSize	23
2.5 CurrentDir	24
2.6 DirAction.....	25
2.7 FileAction.....	27
2.8 FirewallPort.....	29
2.9 FirewallServer	30
2.10 FwAddrType	31
2.11 FwAuthMethods.....	32
2.12 FwPassword	33
2.13 FwSocksVer	34
2.14 FwUsername.....	35
2.15 Host	36
2.16 HostType	37
2.17 Id	38
2.18 LastResult.....	39
2.19 ListType	41
2.20 LocalFile.....	42
2.21 MarkerFrequency	43
2.22 NewName.....	44
2.23 Notify	45
2.24 Passive.....	46
2.25 Password.....	47
2.26 Port	48
2.27 ProxyHost.....	49
2.28 ProxyPassword.....	50
2.29 ProxyPort.....	51
2.30 ProxyType	52
2.31 ProxyUser.....	53
2.32 Quote	54
2.33 RemoteFile	55
2.34 RemoteId	56
2.35 RetryCount	57
2.36 SendData	58
2.37 Target	59
2.38 TransferMode	60
2.39 TransferType	61
2.40 TransmissionMode	62
2.41 UseProperty	63
2.42 User	64
2.43 UseVariant.....	65

2.44	Wildcards	66
3	FTP Client Events	67
3.1	OnClose	67
3.2	OnConnect	68
3.3	OnError	69
3.4	OnList	70
3.5	OnReceive	71
3.6	OnReceiveB	72
3.7	OnReceiveMarker	73
3.8	OnReceiveMarkerB	74
3.9	OnSend	76
3.10	OnSendB	77
3.11	OnTransfer	78
4	FTP Client Methods	79
4.1	Abort	79
4.2	AbortDir	80
4.3	AbortFile	81
4.4	Allocate	82
4.5	AppendFile	83
4.6	Ascii	84
4.7	Binary	85
4.8	BlockMode	86
4.9	Cancel	87
4.10	ChangeDir	88
4.11	Connect	89
4.12	CreateDir	90
4.13	DeleteFile	91
4.14	Disconnect	92
4.15	EventMode	93
4.16	FileMode	94
4.17	FwConnect	95
4.18	GetFile	97
4.19	GetFileWithRestart	98
4.20	GetMultipleFile	99
4.21	ListDir	100
4.22	Login	101
4.23	LongList	102
4.24	ParentDir	103
4.25	Parse	104
4.26	PutFile	105
4.27	PutFileWithRestart	106
4.28	PutMultipleFile	107
4.29	PutUniqueFile	108
4.30	ReceiveB	109
4.31	Reinit	110
4.32	Remote	111
4.33	RemoteAppend	112
4.34	RemoveDir	113
4.35	RenameDir	114
4.36	RenameFile	115
4.37	SendB	116
4.38	ShortList	117
4.39	StreamMode	118

5 Registry Entries 119

5.1 FTP Client Registry Entries..... 119

1 FTP Client ActiveX Overview

1.1 Introduction

The FTP (File Transfer Protocol) is the standard protocol for copying files to and from remote machines. FTP uses separate command and data connections. The Protocol Interpreter (PI) implements the FTP protocol itself, while the Data Transfer Process (DTP) actually performs data transfer. The FTP protocol and the data transfer use entirely separate TCP sessions. With the Distinct FTP client component, you will quickly and securely incorporate automated, timed file transfers, or transfer on demand, capabilities into any application. For example, an application could automatically retrieve customer information from the server when you enter a customer name or identification number. The Distinct FTP client component includes:

- the ability to move entire subtrees of data and to list these.
- support for passive mode.
- firewall proxy (SOCKS 4, SOCKS 5 and HTTP) support to enable its use across intranet firewalls.
- over twenty built in parsers to parse the directory and file information for all the most popular FTPserver types.
- apartment threading, making it easy for you to incorporate ftp functionality that handles more than one concurrent transfer.
- the ability to recover from a failed transfer using the restart option. Note that the server you are connecting to must also have support for this feature.
- real remote to remote file transfer from one FTP server to another FTP server (the file does not buffer on the local system)
- the ability to transfer files in the regular fashion or to transfer them as data directly to an application through the event based file transfer option included in the FTP ActiveX.
- the ability to have more than one user login to a server from the same client without closing the connection.
- the ability to modify the file transfer buffer size to accommodate the network transfer speed.

1.2 Usage of the FTP Client ActiveX Component

See the section entitled "Using Distinct ActiveX controls in various environments" for details on how to add the FTP client control to your project.

Adding FTP functionality to your application should be a fairly straightforward job. Please take a moment to look at the samples provided for your particular programming environment. For FTP we provide a sample that illustrates how to make a ' connection to the FTP server, send FTP commands (the sample shows these as menu items to simplify your search for a particular function) and close your FTP connection. Smaller samples that simply illustrate the Get and Put commands as well as a sample on how to parse the information that is received from the FTP server. This sample gets a long directory listing and parses the information. This sample does not specify the server type being connected to. This means that the default value of the HostType property (Autodetect) is used. Note that if you specifically wish to specify a host type in your application you need to do this by setting the HostType property to one of the supported values. A sample to illustrate how apartment threading can be used to achieve multiple concurrent file transfers is also included.

Once the FTP Client ActiveX control is placed on a form, some properties can be set at design time. For example, if your application is always going to connect to an FTP server that uses a port number other than port 21 which is the default FTP port, you could set this here. You could also set most of the properties required to effect a Get (download) or Put (upload) operation if you are always going to get the same file from the same machine with the same user name. Setting the Host, User, Password, LocalFile and RemoteFile here means that you can achieve your Get or Put operation with about 3 lines of code. If your users will be always connecting through the same HTTP proxy server, these properties can also be set at design time, as can certain properties that are required to connect to an FTP server through a SOCKS firewall. Certain configuration options controlling the type of directory listing and the type of file transfer may never change in a session and need therefore only be set once at design time. Note that when integrating FTP using only properties there are some properties such as Action, DirAction and File Action that would need to be accessed at run time and therefore included in your code.

On the other hand if you wish your built-in FTP functionality to be more interactive with the user, the more efficient way to program this is by using methods. Using methods also becomes indispensable when making use of more advanced functionality such as allowing a transfer to resume following a failure. To reduce your code writing however, you may still wish to set properties such as Port, transmission mode and others at design time. For an interactive session the connection parameters such as host, user and password are usually set at run time through the Connect (or FwConnect) method.

When an FTP connection is established, the OnConnect event will occur before the next line of code is reached. At this point, the user is logged into the FTP server and can now access remote files and directories.

For some users, the local machine is located on a different subnet than the remote host and the only way of communication is through a firewall. If this is the case, then the built-in firewall support of the FTP Client ActiveX control can be used to establish an FTP session via a firewall by setting the Action property to ACTION_FW_CONNECT (or by calling the FwConnect method). In addition to setting the properties for a regular FTP session, the FirewallServer, FirewallPort, FwAddrType, FwAuthMethods, FwUsername and FwPassword properties must also be set before setting the Action property to ACTION_FW_CONNECT. If the connection can be established, the OnConnect event will occur before the next line of code is reached. At this point, the user is logged into the FTP server and can now access remote files and directories as with a regular FTP connection.

The current working directory on the server can be determined by reading the value of the CurrentDir property. The DirAction property (or the AbortDir, ChangeDir, CreateDir, RemoveDir, ListDir, ParentDir and RenameDir methods) is used to move around in the remote directory structure. This property can change the working directory, list its contents and create, delete or rename subdirectories of the current directory. The contents of a remote directory can be listed in short (only the file or subdirectory name) or long (names and attributes) format depending on the setting of the ListType property.

Files in the remote directory can be manipulated with the FileAction property (or with the AbortFile, DeleteFile, GetFile, Putfile, RenameFile, AppendFile, Remote and RemoteAppend methods). Files can be uploaded to the server (put operation), downloaded to the local disk (get operation), deleted or renamed.

The transfer of files can either be event-based or file-based. A file-based transfer means that a file is transferred directly to or from the local disk. A direct file transfer copies files without requiring the application to do any I/O. During an event-based file transfer, the FTP Client control fires events whenever a section of a file has been received or another section of a file can be sent. This allows the application to transfer data to and from storage other than the local disk, such as memory, through DDE or the clipboard. The TransferMode property controls whether a file transfer is event-based or file-based.

Files in the remote directory can also be transferred to another remote directory by setting the FileAction property to FILE_ACTION_REMOTE (or the Remote method) or to FILE_ACTION_REMOTE_APPEND (or the RemoteAppend method). Before setting the Action property, set the RemoteId property to the connection id of the remote machine.

During a direct file based transfer, one or more OnTransfer event will occur to inform the application of the status of the file transfer. This information is useful if the application wishes to display the progress of the file transfer.

Some applications may want to request the FTP Server to listen for a data port and wait for the connection rather than initiate the process when a data transfer request comes in. To achieve this, set the `Passive` property to `True` before a file transfer is initiated.

Any action or command in progress can be aborted by setting the `Action` property to `ACTION_CANCEL` (or by calling the `Cancel` method).

Once a connection is no longer needed the `Disconnect` method needs to be called (or the `Action` property must be set to `ACTION_DISCONNECT`). After the session is disconnected, the `OnClose` event will occur before the next line of code is reached. An application must close all connected sessions before exiting the application.

Dynamic Creation of the Distinct FTP ActiveX control

The FTP Client ActiveX can also be created dynamically in Visual Basic version 6 by declaring it with the `WithEvents` keyword and then creating it with the `New` operator (In this case you can not predefine your properties at design time.) See the section entitled "Using Distinct ActiveX controls in various environments" for details.

Source Code Samples

Several source code samples are provided for the FTP client ActiveX component. Please refer to the section of this manual entitled "How to Run the Sample Applications" for a description of the samples for your programming language.

1.3 FTP Client Property Summary

Account

Optional account name used while connecting to server

Action

Connect, disconnect or abort an FTP session

BinaryData

Contains the binary data received following a call to the RetrieveB method.

BlockSize

Defines the block size for data when transferring in block mode.

CurrentDir

Contains the current working directory on the server

DirAction

Change, create, delete, list or rename directory on server

FileAction

Delete, get, put, append or rename file on server

FirewallPort

Firewall server port number through which a connection can be established.

FirewallServer

Name or dotted decimal internet address of firewall server

FwAddrType

The address format of the destination host

FwAuthMethods

Firewall authentication methods

FwUsername

Firewall username

FwPassword

Firewall password

FwSocksVer

The SOCKS server version

Host

Name of IP address of the FTP server to be connected to.

HostType

Type of server

Id

Unique connection id

LastResult

Result of last command executed by server

ListType

Select short or long directory listing

LocalFile

Path and name of local file to transfer

MarkerFrequency

Frequency of the marker blocks

NewItem

New file or directory name

Notify

Notify number of bytes transferred

Passive

Passive file transfer mode

Password

Password used while connecting to server

Port

FTP service port on server

ProxyHost

Name of proxy server or dotted decimal internet address

ProxyPassword

Password used while connecting through proxy server

ProxyPort

Proxy server port

ProxyType

Type of proxy server

ProxyUser

User name used for proxy server authentication

Quote

Command to send to server

RemoteFile

Name of remote file to transfer

RemoteId

Remote connection id

RetryCount

Number of retries in case of restart command

SendData

Send data buffer

Target

Target of next file or directory action

TransferMode

Select event or file based transfer

TransferType

Select ASCII or binary transfer

TransmissionMode

Mode of transmission

UseProperty

Send data by parameter or property

User

User name used while connecting to server

UseVariant

Send and receive binary or ascii data

Wildcards

Directory wildcards used for directory listing

1.4 FTP Client Event Summary**OnClose**

FTP connection has been closed

OnConnect

FTP connection has been established

OnError

Local error has occurred

OnList

Another line of the directory listing has been received

OnReceive

More data during event based transfer has been received

OnReceiveB

More data during event based transfer has been received as binary

OnReceiveMarker

Marker data during event based transfer has been received

OnReceiveMarkerB

Marker data during event based transfer has been received as binary

OnSend

More data during event based transfer can be sent

OnSendB

More data during event based transfer can be sent as binary

OnTransfer

Number of bytes transferred during file based transfer

1.5 FTP Client Method Summary

Abort

Abort an FTP session

AbortDir

Abort directory listing in progress

AbortFile

Abort file transfer in progress

Allocate

Allocate space on server

AppendFile

Append to a file on server

Ascii

Set transfer type to ASCII file transfer

Binary

Set transfer type to binary file transfer

BlockMode

Set the transmission mode to block mode

Cancel

Abort any outstanding command or action

ChangeDir

Change current working directory on server

Connect

Connect to FTP server

CreateDir

Create directory on server

DeleteFile

Delete file in current directory on server

Disconnect

Close connection to FTP server

EventMode

Set transfer mode to event based transfer

FileMode

Set transfer mode to direct transfer

FwConnect

Establish connection via a firewall

GetFile

Retrieve file from server

GetFileWithRestart

Get the file from server

GetMultipleFile

Get multiple files/directories from the server

ListDir

List contents of current directory on server

Login

Log into the server

LongList

Set listing type to long listing

ParentDir

Change to parent directory on server

Parse

Parse the directory structure

PutFile

Upload file to server

PutFileWithRestart

Put the file on server

PutMultipleFile

Put multiple files/directories on server

PutUniqueFile

Transfer the file from local disk to server and store it as a unique file name

ReceiveB

Reads binary data

Reinit

Terminate the current user

RemoteAppend

Remote to remote file append

Remote

Remote to remote file transfer

RemoveDir

Remove directory on server

RenameDir

Rename directory on server

RenameFile

Rename file in current directory on server

SendB

Send binary data

StreamMode

Set the transmission mode to stream mode

ShortList

Set listing type to short listing

1.6 D_FTP.TXT

The following provides a complete listing of the D_FTP.TXT definition file. If your application uses more than one Distinct ActiveX control in the same form, then some definitions will conflict. For example, the FTP Client ActiveX control includes the definition

```
Global Const ACTION_DISCONNECT = 3
```

in the D_FTP.TXT file and the Telnet ActiveX control includes the definition

```
Global Const ACTION_DISCONNECT = 2
```

in the D_TNET.TXT file. To avoid this conflict, you must rename at least one of the constants (for example, FTP_ACTION_DISCONNECT or TNET_ACTION_DISCONNECT).

```
' FTP Client ActiveX Control
' (C) Copyright 1995 - 1998 by Distinct Corporation
' All rights reserved

' actions
Global Const ACTION_NONE = 0
Global Const ACTION_ABORT = 1
Global Const ACTION_CONNECT = 2
Global Const ACTION_DISCONNECT = 3
Global Const ACTION_FW_CONNECT = 4
Global Const ACTION_CANCEL = 5

Global Const FILE_ACTION_NONE = 0
Global Const FILE_ACTION_ABORT = 1
Global Const FILE_ACTION_DELETE = 2
Global Const FILE_ACTION_GET = 3
Global Const FILE_ACTION_PUT = 4
Global Const FILE_ACTION_RENAME = 5
Global Const FILE_ACTION_APPEND = 6
Global Const FILE_ACTION_REMOTE = 7
Global Const FILE_ACTION_REMOTE_APPEND = 8

Global Const DIR_ACTION_NONE = 0
Global Const DIR_ACTION_ABORT = 1
Global Const DIR_ACTION_CHANGE = 2
Global Const DIR_ACTION_CREATE = 3
Global Const DIR_ACTION_DELETE = 4
Global Const DIR_ACTION_LIST = 5
Global Const DIR_ACTION_PARENT = 6
Global Const DIR_ACTION_RENAME = 7

' modes
Global Const TRANSFER_TYPE_ASCII = 0
Global Const TRANSFER_TYPE_BINARY = 1

Global Const TRANSFER_MODE_FILE = 0
Global Const TRANSFER_MODE_EVENT = 1

Global Const LIST_TYPE_SHORT = 0
Global Const LIST_TYPE_LONG = 1

Global Const PROXY_TYPE_NONE = 0
Global Const PROXY_TYPE_HTTP = 1

Global Const TRANSMISSION_MODE_STREAM = 0
Global Const TRANSMISSION_MODE_BLOCK = 1

' the address type of the destination host
Global Const FW_ADDR_IP4 = 1
Global Const FW_ADDR_DNS = 3
Global Const FW_ADDR_IP6 = 4

'the firewall server version
Global Const FW_VERSION5 = 2
Global Const FW_VERSION4 = 4
```

```
' result
Global Const FTP_OK = 0
Global Const FTP_ERROR = 1
Global Const FTP_CONN_CLOSED = 3
Global Const FTP_NO_ENTRY = -1
Global Const FTP_BAD_FILE_TYPE = -2
Global Const FTP_NOT_CONNECTED = -3
Global Const FTP_BAD_ARGUMENT = -4
Global Const FTP_BAD_COMMAND = -5
Global Const FTP_FILE_ERROR = -6
Global Const FTP_DATA_CONN_ERR = -7
Global Const FTP_ACCEPT_ERR = -8
Global Const FTP_REPLY_TIMEOUT = -10
Global Const FTP_ACCEPT_TIMEOUT = -11
Global Const FTP_SEND_TIMEOUT = -12
Global Const FTP_CANCELLED = -13
Global Const FTP_BUSY = -14
Global Const FTP_DATA_TIMEOUT = -15
Global Const FTP_PASSIVE_FAILED = -16
Global Const FTP_CONNECT_ERR = -17
Global Const FTP_FIREWALL_ERR = -18
Global Const FTP_DSTNCT32_ERROR = 1000
Global Const FTP_WSASTARTUP_ERROR = 1001
Global Const FTP_SYSTEM_ERROR = 1002
Global Const FTP_INVALID_HOSTNAME = 1003
Global Const FTP_SOCKET_ERROR = 1004
Global Const FTP_BIND_ERROR = 1005
Global Const FTP_CONNECT_ERROR = 1006
Global Const FTP_ASYNC_ERROR = 1007
Global Const FTP_MAX_CONNECTIONS = 1008
Global Const FTP_OOB_ERROR = 1009
```

```
' error codes
Global Const ERR_CANNOT_CHANGE_XFER_TYPE = 1
Global Const ERR_CANNOT_CHANGE_XFER_MODE = 2
Global Const ERR_CANNOT_CHANGE_LIST_TYPE = 3
Global Const ERR_CANNOT_CHANGE_PORT = 4
Global Const ERR_PORT_UNDEFINED = 5
Global Const ERR_HOST_UNDEFINED = 6
Global Const ERR_IN_TRANSFER = 7
Global Const ERR_CANNOT_CONNECT = 8
Global Const ERR_NEED_ACCOUNT = 9
Global Const ERR_CANNOT_LOG_IN = 10
Global Const ERR_NOT_CONNECTED = 11
Global Const ERR_IN_ACTION = 12
Global Const ERR_NO_DIR_TARGET = 13
Global Const ERR_NO_NEW_DIR_NAME = 14
Global Const ERR_NO_FILE_TARGET = 15
Global Const ERR_NO_NEW_FILE_NAME = 16
Global Const ERR_NO_REMOTE_FILE = 17
Global Const ERR_NO_LOCAL_FILE = 18
Global Const ERR_CANNOT_OPEN_LOCAL_FILE = 19
Global Const ERR_UNABLE_TO_LOAD = 20
Global Const ERR_FW_SERVER_NOT_DEFINED = 21
Global Const ERR_CANNOT_CHANGE_PROXY_TYPE = 22
Global Const ERR_CANNOT_CHANGE_TRANSMISSION_MODE = 23
Global Const ERR_NO_LOCAL_DIR = 24
Global Const ERR_CANNOT_CREATE_MARKER_FILE = 25
Global Const ERR_CANNOT_CHANGE_HOST_TYPE = 26
Global Const ERR_CANNOT_CHANGE_BLOCK_SIZE = 27
Global Const ERR_CANNOT_CHANGE_MARKER_FREQUENCY = 28
Global Const ERR_CANNOT_CHANGE_RETRY_COUNT = 29
Global Const ERR_NO_LIST_LINE = 30

'server types
Global Const SERVER_AUTODETECT = 0
Global Const SERVER_UNIX = 1
Global Const SERVER_ULTRIX = 2
Global Const SERVER_VMS_UCX = 3
Global Const SERVER_DISTINCT = 4
Global Const SERVER_FTP_DOS = 5
Global Const SERVER_DOS = 6
Global Const SERVER_IBM_VM = 7
Global Const SERVER_IBM_AIX = 8
Global Const SERVER_VMS_MULTINET = 9
Global Const SERVER_VMS_FUSION = 10
Global Const SERVER_VMS_WGONG = 11
Global Const SERVER_VMS_UCX_2 = 12
Global Const SERVER_IBM_MVS_2 = 13
Global Const SERVER_IBM_MVS_3 = 14
Global Const SERVER_IBM_OS2 = 15
Global Const SERVER_VMS_PROCESS = 16
Global Const SERVER_IBM_AS_400 = 17
Global Const SERVER_IBM_MVS_4 = 18
Global Const SERVER_IBM_AS_400_3 = 19
Global Const SERVER_WINDOWS_NT = 20
```


2. FTP Client Properties

2.1 Account

Summary

Optional account name used while connecting to server.

Description

The Account property is used during the login process to the server. Some FTP servers require that an account name be specified in addition to the user name and the password. The majority of servers, however, do not require this and in these cases this property can be left blank.

If an application prompts the user for the user name and password before connecting to the server, then adding fields to allow the user to enter an optional account name and an FTP server port (see the Port property reference page) should be provided. If the login parameters are specified at design time, then the local system administrator should be consulted to determine if an account name or a port other than the default port is required.

This property can be changed at design time and at run time before a connection has been established. There is no default value for this property.

Example

```
FTPClient.Host = "speedy"  
FTPClient.User = "santa"  
FTPClient.Password = "north pole"  
FTPClient.Account = "sc001d"  
FTPClient.Action = ACTION_CONNECT
```

2.2 Action

Summary

Connect, disconnect or abort an FTP session.

Description

The Action property controls the connection state of the FTP Client ActiveX control. An FTP session can be established, closed or aborted by assigning one of the following values to the property.

Value	Meaning
ACTION_ABORT	Abort session.
ACTION_CONNECT	Establish session.
ACTION_DISCONNECT	Close session.
ACTION_FW_CONNECT	Establish session via a firewall.
ACTION_CANCEL	Cancel any action.

This property can be changed at run time only.

Before setting the Action property to ACTION_CONNECT, the following properties must be initialized. The Host property must be set to the name or internet address (in dotted decimal notation) of the FTP server. The Port property must be set to the remote port on which the FTP service is running (most servers use the default FTP service port of 21). The User and Password properties must contain a valid user name and password to complete the login process. Some servers also require that the Account property contain the name of an account for the user. Most applications will prompt the user with a dialog box for the information to place into these properties, but all of them can also be specified at design time.

If the FTP Client machine is located on a different subnet than the remote FTP Server machine and the only form of communication between these two machines is through a firewall gateway, then the built-in firewall support of the FTP Client ActiveX control can be used to establish an FTP session. To establish a connection with a remote FTP Server through a firewall, set the Action property to ACTION_FW_CONNECT. Before setting the Action property to ACTION_FW_CONNECT, the following properties must be initialized. The FirewallServer property must be set to the name or internet address (in dotted decimal notation) of the firewall server. The FirewallPort property must be set to the firewall service port. In addition to the FirewallServer and FirewallPort properties, the Host, User, Port, Password, and Account properties must also be set as mentioned above. Depending on the type of address specified in the Host property the FwAddrType property must be accordingly set, if the Host property contains the IP address of the remote host then the FwAddrType must be set to FW_ADDR_IP4 and if contains a machine name then the FwAddrType property must contain FW_ADDR_DNS. Distinct FTP ActiveX Control supports both SOCKS version 5 and SOCKS version 4, the application can specify the SOCKS version in the FwSocksVer property. If the SOCKS version is 5 then the application can specify a authentication method in the FwAuthMethods property, currently only the Username/Password (FwUsername and FwPassword) authentication protocol is supported.

If the connection can be established, then the OnConnect event will be fired. If the connection cannot be established, then the OnError event will be fired. These events will occur before the next statement (i.e. the statement following the assignment of ACTION_CONNECT to the Action property) is executed. The application should set a flag in the OnConnect and OnError events, so that it can determine if the session has been established or not. In addition, the application may want to display an error message in the OnError event to inform the user that the connection has not been established. Please check the reference page of the OnError event for a complete listing of error codes.

While an action is in progress, it can be canceled by setting the Action property to ACTION_CANCEL. Actions that can be aborted include trying to establish an FTP session, transferring a file, listing a directory, and any directory or file manipulations.

Once a connection is no longer needed, the session can be terminated by setting the Action property to ACTION_DISCONNECT. An application must close all connections it has created before it quits. A connection can also be closed by setting the Action property to ACTION_ABORT. This action resets and closes the connection without properly closing down and should not be called under normal circumstances.

The Connect, Disconnect, Abort, FwConnect, and Cancel methods accomplish the same as the above actions. Please check the reference pages of these methods for more detailed information on their usage.

The value of the LastResult property can be checked to determine if the action has been performed successfully. There is no default value for this property.

Example

```
FTPClient.Host = "speedy.distinct.com"  
FTPClient.User = "santa"  
FTPClient.Password = "north pole"  
FTPClient.Account = "sc001d"  
FTPClient.Action = ACTION_CONNECT
```

Note

If you are writing a new application it is recommended to use the corresponding methods for these actions instead.

2.3 BinaryData

Summary

Contains the binary data received following a call to the ReceiveB method.

Description

After each call to the ReceiveB method in a C# application this property needs to be read to access binary data.

Example

```
Private void Ftp_OnReceive (object sender, System.EventArgs e)
{
    int len;
    int bytes;
    Object arrData = new byte[bytes];

    Len = axFtp1.ReceiveB (arrData);
    If (len > 0)
    {
        object pBuf = new Byte[len];
        pBuf = axFtp1.BinaryData;
        byte []RcvByte = (byte [])pBuf;
        .....
        .....
    }
}
```

2.4 BlockSize

Summary

Specifies the size of each block of data transferred in a block-mode file transfer.

Description

The BlockSize property controls the block size during data transfer in block mode. BlockSize is a short integer. The BlockMode method should be called after setting this property to set the mode and block size.

Block Mode may not be supported by all FTP servers.

This property can be changed at run time as well as design time. The default value for this property is 4096.

Example

```
FTPClient.BlockSize = 8192  
FTPClient.BlockMode()
```

2.5 CurrentDir

Summary

Contains the current working directory on the server.

Description

The CurrentDir property is used to get the complete path of the current working directory on the FTP server. A variable must be assigned to save the value of this property and must be updated each time after changing the current directory. The value can only be accessed while the connection is open.

This property is usually read once after establishing a connection and every time after changing the remote working directory. An application could, for example, update the current directory display in a text control after every directory operation.

This property can only be read at run time while a connection is open. There is no default value for this property.

Example

```
Dim Path As String
```

```
FTPClient.Action = ACTION_CONNECT  
Path = FTPClient.CurrentDir
```

2.6 DirAction

Summary

The DirAction property allows you to change (or change to the parent), create, delete, list or rename directory on server.

Description

The DirAction property accesses directories on the FTP server. Subdirectories in the current working directory can be created, deleted and renamed and the current working directory can be changed or its contents can be listed in short or long format.

Value	Meaning
DIR_ACTION_ABORT	Abort directory list action in progress.
DIR_ACTION_CHANGE	Change current working directory.
DIR_ACTION_CREATE	Create subdirectory in current directory.
DIR_ACTION_DELETE	Delete subdirectory in current directory.
DIR_ACTION_LIST	List contents of current directory.
DIR_ACTION_PARENT	Change to parent directory of current directory.
DIR_ACTION_RENAME	Rename subdirectory in current directory.

This property can be changed at run time only.

Before setting the DirAction property to DIR_ACTION_CREATE, DIR_ACTION_DELETE or DIR_ACTION_RENAME, the name of the subdirectory on which the operation should be performed must be assigned to the Target property. In case of the DIR_ACTION_RENAME action, the new name of the subdirectory must also be assigned to the NewName property.

Changing the current working directory with the DIR_ACTION_PARENT action does not require any other properties. If the current directory is changed with the DIR_ACTION_CHANGE action, then the Target property must contain the name of the directory to change to. This target directory can be a subdirectory of the current directory, ".." (identical to DIR_ACTION_PARENT) or a fully qualified path to a new working directory. This allows an application to change to any directory on the server in just one step.

The contents of the current remote directory can be listed by setting DirAction to DIR_ACTION_LIST. The setting of the ListType property determines if a short or a long listing will be sent. For every entry in the remote directory, the OnList event is fired. If an application does not want to receive any more entries, then it can set the DirAction property to DIR_ACTION_ABORT in response to this event. Please check the OnList reference page of more information.

The value of the LastResult property can be checked to determine if the action has been performed successfully.

The AbortDir, ChangeDir, CreateDir, RemoveDir, ListDir, ParentDir and RenameDir methods accomplish the same as the above actions. Please check the reference pages of these methods for more detailed information on their usage.

There is no default value for this property.

Example

```
FTPClient.ListType = LIST_TYPE_LONG
FTPClient.DirAction = DIR_ACTION_LIST
```

Note

If you are writing a new application it is recommended to use the corresponding methods for these actions instead.

2.7 FileAction

Summary

The FileAction property is used to delete, get, put, append to or rename a file on the FTP server.

Description

The FileAction property accesses files in the current working directory of the FTP server. Files can be transferred to the server (put), downloaded to the local disk (get), appended, deleted or renamed.

Value	Meaning
FILE_ACTION_ABORT	Abort file action in progress.
FILE_ACTION_DELETE	Delete file in current directory.
FILE_ACTION_GET	Retrieve file from server.
FILE_ACTION_PUT	Upload file to server.
FILE_ACTION_RENAME	Rename file in current directory.
FILE_ACTION_APPEND	Append to a file on the server.
FILE_ACTION_REMOTE	Upload file on remote server to another remote server.
FILE_ACTION_REMOTE_APPEND	Append file on remote server to another remote server

This property can be changed at run time only.

Before setting the FileAction property to FILE_ACTION_DELETE or FILE_ACTION_RENAME, the name of the file on which the operation should be performed must be assigned to the Target property. In case of the FILE_ACTION_RENAME action, the new name of the file must also be assigned to the NewName property.

Files can be transferred to and from the FTP server either directly or through events, depending on the setting of the TransferMode property. In direct mode, a FILE_ACTION_PUT operation transfers the local file specified with the LocalFile property to the remote file specified with the RemoteFile property and a FILE_ACTION_APPEND operation will append the contents of the local file specified with the LocalFile property to the remote file specified with the RemoteFile property. A FILE_ACTION_GET operation, in the direct mode, transfers the remote file specified with the RemoteFile property to the local file specified with the LocalFile property. This feature allows an application to transfer complete files of any size without doing any I/O itself. The LocalFile property must contain a fully qualified path name since there is no current local directory.

During a direct file transfer, one or more OnTransfer events will occur to notify the application of the status of the file transfer (number of bytes copied). The application can use this information to display the progress of the file transfer. This event will be fired only if the Notify property is set to True.

When a file is transferred through events, then one or more OnSend events (in case of a put or an append operation) or one or more OnReceive events (in case of a get operation) will be fired. An OnSend event asks the application to provide up to a maximum number of bytes of data, which are to be saved in the remote file. OnSend events are generated until the application indicates that there is no more data to send. An OnReceive event delivers a portion of the data from the remote file to the application. Event based file transfer allows an application to move data to and from storage other than the local disk, such as through DDE or the clipboard. For event based file transfer, the LocalFile property is ignored and can be left blank.

Files can be transferred from the FTP server to another FTP server via a FILE_ACTION_REMOTE operation. Use the FILE_ACTION_REMOTE_APPEND operation to append the contents of a file on the FTP server, to a file on another FTP server. Before setting the FileAction property to

FILE_ACTION_REMOTE or FILE_ACTION_REMOTE_APPEND, set the LocalFile property to the file on the FTP server and set the RemoteFile property to the file name on the other FTP server. The RemoteId property must be set to the connection id of the other FTP client involved in the remote to remote file transfer.

If a file should be transferred in its original format without any filtering, then the TransferType property must be set to TRANSFER_TYPE_BINARY. If the transfer type is set to TRANSFER_TYPE_ASCII, then carriage return and line feed pairs are converted to line feeds during a put operation and line feeds are converted to carriage return and line feed pairs during a get operation. The ASCII mode conversion is normally used to convert text files to a format suitable for text editors on the destination machine.

Setting the FileAction property to FILE_ACTION_ABORT will abort any get, put or append operation in progress. This is usually done during an OnSend or OnReceive event to make sure that no further such events will occur.

The value of the LastResult property can be checked to determine if the action has been performed successfully.

The AbortFile, DeleteFile, GetFile, PutFile, RenameFile, AppendFile, Remote and RemoteAppend methods accomplish the same as the above actions. Please check the reference pages of these methods for more detailed information on their usage.

There is no default value for this property.

Example

```
FTPClient.RemoteFile = "test.exe"  
FTPClient.LocalFile = "c:\test.exe"  
FTPClient.FileAction = FILE_ACTION_GET
```

Note

If you are writing a new application it is recommended to use the corresponding methods for these actions instead.

2.8 FirewallPort

Summary

Specifies the SOCKS firewall server port number.

Description

The FirewallPort property specifies the port on the firewall server through which a connection can be established. The FirewallPort property must be set before a connection is established with the remote FTP server (by setting the Action property to ACTION_FW_CONNECT or by calling the FwConnect method).

This property can be changed at any time except after a connection has been established with the remote FTP server (by setting the Action property to ACTION_CONNECT or ACTION_FW_CONNECT). The default value for this property is 1080.

Example

```
FTPClient.FirewallServer = "127.43.101.10"  
FTPClient.FirewallPort = 1080  
FTPClient.Host = "127.43.101.12"  
FTPClient.Action = ACTION_FW_CONNECT
```

Note

This property is used only if you are using the Action property instead of the FwConnect method.

2.9 FirewallServer

Summary

Specifies the name of the firewall server or dotted decimal internet address.

Description

The FirewallServer property specifies the name or internet address of a firewall through which a connection is to be made. This property must be set before a connection can be established (by changing the Action property). There are three possible ways of specifying a firewall server name.

Machine Name

An application only needs to specify the name of the firewall server if the host is located on the same network as the local PC or if the internet address of the host is defined in the local hosts data base. If the firewall server is not on the local network, then the underlying protocol will route the traffic through a gateway. If the host is not defined in the local host's database, then the underlying protocol will contact the name server to resolve the internet address of the firewall server.

Machine and Domain Name

An application needs to specify the machine name and the domain name if the host is not located on the same network as the local PC. Fully domain qualified machine names are written from left to right in ascending order (for example, *speedy.distinct.com*). If both machine and domain names are specified, then the underlying protocol will contact the name server to resolve the internet address of the firewall server.

Internet Address

Sometimes the user knows only the internet address of the firewall server that he or she wants to use. In this case, the internet address can be entered in what is known as the dotted decimal notation (for example, *127.43.101.12*). If the host identified by this address is not on the local network, then the underlying protocol will route the traffic through a gateway.

This property can be changed at design time and at run time before a connection has been established. There is no default value for this property.

Example

```
FTPClient.FirewallServer = "127.43.101.10"  
FTPClient.FirewallPort = 1080  
FTPClient.Host = "127.43.101.12"  
FTPClient.Action = ACTION_FW_CONNECT
```

Note

This property is used only if you are using the Action property instead of the FwConnect method

2.10 FwAddrType

Summary

Specifies the type of address that you wish to connect to through the firewall server.

Description

The *FwAddrType* property specifies the address type in the *Host* property. This property must be set before a connection can be established by calling the *FwConnect* method (or by changing the *Action* property to ACTION_FW_). There are three possible ways of specifying a destination address in the *Host* Property and therefore the *FwAddrType* property can have three possible formats:

FW_ADDR_IP4	Address is a version 4 IP address
FW_ADDR_DNS	Address is a DNS style domain name
FW_ADDR_IP6	Address is a version 6 IP address

The *FwAddrType* property must be set to FW_ADDR_IP4 if the application is specifying a Internet address in the dotted decimal notation (**198.211.122.133**) in the *Host* property. If the application wants to specify the machine name or machine name and domain name (for example **speedy.distinct.com**) as the *Host* they must set the *FwAddrType* property to FW_ADDR_DNS.

Note that the FW_ADDR_IP6 is not currently supported.

This property can be changed at design time and at run time before a connection has been established.

The default value for this property is FW_ADDR_IP4.

Example

```
Ftp.FwAddrType = FW_ADDR_DNS
Result = Ftp.FwConnect ("sparky.distinct.com", "1080", "speedy.distinct.com", "santa", "north
pole", "")
If Result = False Then
MsgBox "Unable to connect to server via a firewall", 64, "Sample Program"
End If
```

2.11 FwAuthMethods

Summary

Specifies the authentication method used when connecting through a firewall.

Description

The *FwAuthMethods* property specifies the authentication method that can be used when connecting to the firewall server. This property must be set before a connection can be established by calling the *FwConnect* method (or set the *Action* property is set to ACTION_FW_CONNECT). The *FwAuthMethods* property can be "0", "1" or "2" or a combination of "0", "1", "2", for example "01" or "12". "0" means that no authentication is required, "1" means GSSAPI and "2" means that a valid username and password is required. Currently only methods "0" and "2" are supported.

This property can be changed at design time or at run time before a connection has been established.

The default value for this property is "0".

Example

```
Ftp.FwAddrType = FW_ADDR_DNS
Ftp.FwAuthMethods = "2"
Ftp.FwUsername = "joe"
Ftp.FwPassword = "distinct"
Result = Ftp.FwConnect ("sparky.distinct.com", "1080", "speedy.distinct.com", "santa", "north
pole", "")
If Result = False Then
    MsgBox "Unable to connect to server via a firewall", 64, "Sample Program"
End If
```

2.12 FwPassword

Summary

Specifies the password for the firewall authentication.

Description

The *FwPassword* property specifies the valid password required during authentication when establishing a connection via a firewall. A valid password is required when connecting to SOCKS version 5 server if the authentication method (*FwAuthMethods*) specified is "2" (Username/Password authentication protocol).

This property can be changed at design time and at run time before a connection has been established by calling the *FwConnect* method (or setting the *Action* property to *ACTION_FW_CONNECT*).

The property does not have any default value.

Example

```
Ftp.FwAddrType = FW_ADDR_DNS
Ftp.FwAuthMethods = "2"
Ftp.FwUsername = "joe"
Ftp.FwPassword = "distinct"
Result = Ftp.FwConnect ("sparky.distinct.com", "1080", "speedy.distinct.com", "santa", "north
pole", "")
If Result = False Then
    MsgBox "Unable to connect to server via a firewall", 64, "Sample Program"
End If
```

2.13 FwSocksVer

Summary

Allows the user to specify the SOCKS version.

Description

The *FwSocksVer* property is used specify the version of the SOCKS server. This property must be set before establishing a connection via a firewall (by calling the *FwConnect* method or setting the *Action* property to ACTION_FW_CONNECT)

This property can have any one or a combination of the following values.

Value	Meaning
FW_VERSION5	The SOCKS version is 5.
FW_VERSION4	The SOCKS version is 4

If the firewall server version is unknown then the application can specify both FW_VERSION5 and FW_VERSION4. The Distinct FTP ActiveX control will automatically detect the firewall server version and make the appropriate connection.

This property can be set at design time or at run time before a connection is established by calling the *FwConnect* method or setting the *Action* property to ACTION_FW_CONNECT.

Example

```
Ftp.FwAddrType = FW_ADDR_DNS
Ftp.FwAuthMethods = "2"
Ftp.FwUsername = "joe"
Ftp.FwPassword = "distinct"
Ftp.FwSocksVer = FW_VERSION5
Result = Ftp.FwConnect ("sparky.distinct.com", "1080", "speedy.distinct.com", "santa", "north
pole", "")
If Result = False Then
    MsgBox "Unable to connect to server via a firewall", 64, "Sample Program"
End If
```

2.14 FwUsername

Summary

Specifies the username for the firewall authentication.

Description

The *FwUsername* property specifies a valid username when establishing a connection via a firewall. This property must be set before calling the *FwConnect* method (or setting the *Action* property to *ACTION_FW_CONNECT*.)

A valid username is mandatory when a connection needs to be established with a SOCKS version 4 server. It is also required when connecting to a SOCKS version 5 server if the authentication method (*FwAuthMethods*) specified is "2" (Username/Password authentication protocol).

This property can be changed at design time and at run time before a connection has been established by setting the *Action* property to *ACTION_FW_CONNECT* or by calling the method *FwConnect*.

The property does not have any default value.

Example

```
Ftp.FwAddrType = FW_ADDR_DNS
Ftp.FwAuthMethods = "2"
Ftp.FwUsername = "joe"
Ftp.FwPassword = "distinct"
Result = Ftp.FwConnect ("sparky.distinct.com", "1080", "speedy.distinct.com", "santa", "north
pole", "")
If Result = False Then
    MsgBox "Unable to connect to server via a firewall", 64, "Sample Program"
End If
```

2.15 Host

Summary

Specifies the name or dotted decimal internet address of the FTP server that the FTP client will connect to.

Description

The Host property specifies the name or internet address of an FTP server. This property must be set before a session can be established. There are three possible ways of specifying an FTP server.

Machine Name

An application only needs to specify the name of the FTP server if the server is located on the same network as the local PC or if the internet address of the server is defined in the local host table. If the FTP server is not on the local network, then the underlying protocol will route the traffic through a gateway. If the FTP server is not defined in the local host table, then the underlying protocol will contact the domain server to resolve the internet address of the server.

Machine and Domain Name

An application needs to specify the machine name and the domain name if the FTP server is not located on the same network as the local PC. Fully domain qualified machine names are written from left to right in ascending order (for example, *speedy.distinct.com*). If both machine and domain names are specified, then the underlying protocol will contact the domain server to resolve the internet address of the server.

Internet Address

Sometimes the user knows only the internet address of the FTP server that he or she wants to use. In this case, the internet address can be entered in what is known as the dotted decimal notation (for example, *127.43.101.12*). If the FTP server identified by this address is not on the local network, then the underlying protocol will route the traffic through a gateway.

This property can be changed at design time and at run time before a connection has been established. There is no default value for this property.

Example

```
FTPClient.Host = "127.43.101.12"  
FTPClient.User = "santa"  
FTPClient.Password = "north pole"  
FTPClient.Account = ""  
FTPClient.Action = ACTION_CONNECT
```

Note

This property is only used if you are using the Action property instead of the Connect or FwConnect methods.

2.16 HostType

Summary

Specifies the type of FTP server that the client will be connecting to.

Description

The HostType property is used to set the host type of the remote machine. The HostType is used when parsing the server reply, and during multiple get file operations.

The default value is SERVER_AUTODETECT. If the automatic detection fails the host type will default to SERVER_UNIX. This property can be changed at run time only.

If the remote machine does not support the SYST command to determine the type of server, then this property should be set to the type of the FTP server.

The following values are supported:

Value	Meaning
SERVER_AUTODETECT	The FTP Client ActiveX will automatically detect the server type.
SERVER_UNIX	Unix FTP Server
SERVER_LINUX	Linux FTP Server
SERVER_ULTRIX	Ultrix FTP Server
SERVER_VMS_UCX	VMS UCX FTP Server
SERVER_DISTINCT	Distinct FTP Server
SERVER_FTP_DOS	FTP Software DOS FTP Server
SERVER_DOS	DOS FTP Server
SERVER_IBM_VM	IBM VM FTP Server
SERVER_IBM_AIX	IBM AIX FTP Server
SERVER_VMS_MULTINET	DEC VMS TGV Multinet FTP Server
SERVER_VMS_FUSION	DEC VMS FTP Server
SERVER_VMS_WGONG	DEC VMS Wollongong FTP Server
SERVER_VMS_UCX_2	VMS UCX Version 2 FTP Server
SERVER_IBM_MVS_2	IBM MVS Version 2 FTP Server
SERVER_IBM_MVS_3	IBM MVS Version 3 FTP Server
SERVER_IBM_OS2	IBM OS/2 FTP Server
SERVER_VMS_PROCESS	DEC VMS Process TCPWare FTP Server
SERVER_IBM_AS_400	IBM AS 400 FTP Server
SERVER_IBM_MVS_4	IBM MVS Version 4 FTP Server
SERVER_IBM_AS_400_3	IBM AS 400 Version 3 FTP Server
SERVER_WINDOWS_NT	Windows NT FTP Server

Example

```
FTPClient.HostType = SERVER_DISTINCT
FtpClient.Connect (ftp.distinct.com, "santa", "northpole", "")
If Result = False Then
    MsgBox "Unable to connect to server", 64, "Sample Program"
End if.
```

2.17 Id

Summary

Unique connection id.

Description

The Id property specifies a unique connection id for the FTP client. This Id will be different for each connection established through the ActiveX control.

This id can be used as the RemoteId of another FTP Client to transfer files remotely. Remote to remote file transfers can then be initiated by the other FTP client by calling the Remote/RemoteAppend method (or setting its FileAction property to FILE_ACTION_REMOTE or FILE_ACTION_REMOTE_APPEND). See the documentation on RemoteId, Remote and RemoteAppend for more information on remote to remote file transfer.

The id can be retrieved after a successful connection (i.e. After setting the Action property to ACTION_CONNECT, or calling the Connect method). There is no default value for this property.

Example

```
Result = FtpClient2.Remote (FtpClient.Id, "test, "abc")
If Result = False Then
    MsgBox "Cannot perform the specified remote to remote operation", 64, "Sample Program"
End If
```

2.18 LastResult

Summary

Contains the last reply code from the server

Description

The LastResult property contains the result of the last operation on the FTP server in response to the most recent client request. The property can have any one of the following values.

Value	Meaning
FTP_OK	Operation completed successfully.
FTP_ERROR	Operation could not be completed.
FTP_CONN_CLOSED	Connection was closed by server.
FTP_NO_ENTRY	Too many FTP connections open.
FTP_BAD_FILE_TYPE	File type is incorrect.
FTP_NOT_CONNECTED	Connection was reset.
FTP_BAD_ARGUMENT	Incorrect argument specified.
FTP_BAD_COMMAND	Incorrect command specified.
FTP_FILE_ERROR	Error during file transfer.
FTP_DATA_CONN_ERR	Error opening data connection.
FTP_ACCEPT_ERR	Error accepting data connection.
FTP_REPLY_TIMEOUT	Timed out waiting for a reply.
FTP_ACCEPT_TIMEOUT	Timed out waiting for data connection.
FTP_SEND_TIMEOUT	Timed out during send operation.
FTP_CANCELLED	Transfer aborted.
FTP_BUSY	FTP server is busy.
FTP_DATA_TIMEOUT	Timed out waiting for data.
FTP_INVALID_HOSTNAME	Specified host name is not valid.
FTP_CONNECT_ERROR	Error in connecting to remote host.
FTP_ASYNC_ERROR	Error in enabling asynchronous notification.
FTP_MAX_CONNECTIONS	Cannot open any more connections.
FTP_OOB_ERROR	Error in handling urgent data options for the control port.

If a different value than those listed above appears, then an error occurred while the server was executing the command. The LastResult property will contain the reply code from the FTP server. Some of the reply codes are listed below. For more information about the reply codes, please refer to RFC 959 - "File Transfer Protocol (FTP)".

Value	Meaning
500	Syntax error, command unrecognized, command line too long.
501	Syntax error in parameters or arguments.
502	Command not implemented.
503	Bad sequence of commands.
504	Command not implemented for that parameter.
530	Not logged in.
532	Need account for storing files.
550	Requested action not taken (file not found, no access).
551	Requested action aborted (page type unknown).
552	Requested file action aborted (exceeded storage allocation).
553	Requested action not taken (file name not allowed).

The LastResult property reflects the result of the last operation caused by setting the Action, DirAction, FileAction or TransferType (if connected) properties (or by calling the equivalent method).

The value of this property should be checked immediately after each operation. Calling other methods or setting other properties may change the value of the property.

This property can be read at any time. There is no default value for this property.

Example

```
FTPClient.RemoteFile = "test.exe"  
FTPClient.LocalFile = "c:\test.exe"  
FTPClient.FileAction = FILE_ACTION_GET  
If FTPClient.LastResult <> FTP_OK Then  
    MsgBox "Unable to transfer file", 64, "Sample Program"  
End If
```

2.19 ListType

Summary

Specifies whether a short or long directory listing should be generated

Description

The ListType property determines what type of directory listing will be generated. It can be set to either one of the following two values.

Value	Meaning
LIST_TYPE_SHORT	File or directory name only.
LIST_TYPE_LONG	File or directory name plus attributes.

Directory listings are generated by setting the DirAction property to DIR_ACTION_LIST. Until all files in the current remote working directory have been listed (or until the DirAction property is set to DIR_ACTION_ABORT), OnList events are generated for each file or directory in the remote directory. OnList events generated for a short listing will only include the name of a subdirectory or of a file, but long listing lines may include any number of additional pieces of information. Most servers report the size of a file, the time and date of the last modification and the file attributes in a long listing.

The Wildcards property can be used to restrict the directory listing to only those files and directories which match a given wildcard.

The format of long directory listing entries varies from one FTP server to another. Fields, such as name, size, date and time, may be displayed differently and may not be in the same order. It is up to the application to interpret the incoming lines. In most cases, the directory entry lines can simply be displayed to the user without any further processing.

The ShortList and LongList methods can also be used to set the type of listing. Please check the reference pages of these methods for more detailed information on their usage.

Example

```
FTPClient.ListType = LIST_TYPE_SHORT  
FTPClient.DirAction = DIR_ACTION_LIST
```

Note

If you are writing a new application it is recommended to use the corresponding methods for these actions instead. These are the ShortList and LongList methods.

2.20 LocalFile

Summary

Specifies the complete path and name of the local file to be transferred in a direct (file-based) file transfer.

Description

The LocalFile property specifies the name of the local file to be used during a direct file transfer. File transfers are initiated by setting the FileAction property to FILE_ACTION_GET or FILE_ACTION_PUT. A file transfer is direct rather than event based if the TransferMode property is set to TRANSFER_MODE_FILE rather than TRANSFER_MODE_EVENT.

During an event-based file transfer, the LocalFile property is ignored and can be left blank.

The local file must be specified with a fully qualified path (such as "C:\MYDIR\TEST.TXT"). Since there is no local current working directory, a file specified without a path may reside in the Windows directory, in the directory from which the application was started or in the directory defined as the application's working directory.

This property can be changed at any time except during a file transfer. There is no default value for this property.

Example

```
FTPClient.RemoteFile = "test.exe"  
FTPClient.LocalFile = "c:\test.exe"  
FTPClient.FileAction = FILE_ACTION_GET
```

Note

For new applications specify the source or destination in the GetFile, PutFile, Remote and RemoteAppend methods instead of setting this property.

2.21 MarkerFrequency

Summary

Specifies the frequency of a marker block in a block mode file transfer.

Description

The MarkerFrequency property sets the frequency of marker blocks during a file transfer. For example, if MarkerFrequency = 2, one marker block will be sent after every two data blocks.

Set this property to allow file transfer recovery following a transfer failure, for large file transfers. The transfer mode must be Block mode for this property to have any effect.

This property can be changed at run time as well as design time. Default value for this property is 0, which means that it is off.

Example

```
FTPClient.MarkerFrequency = 2  
FTPClient.BlockSize = 8192  
FTPClient.BlockMode()
```

See Also

GetFileWithRestart and PutFileWithRestart methods; OnReceiveMarker and OnReceiveMarkerB events.

2.22 NewName

Summary

Specifies the new file or directory name in a rename operation.

Description

The NewName property is used to specify the new name of a file or a subdirectory during a rename operation. Files are renamed by setting the FileAction property to FILE_ACTION_RENAME and subdirectories are renamed by setting the DirAction property to DIR_ACTION_RENAME.

The file or subdirectory to be renamed must be identified by the Target property.

This property can be changed at any time. There is no default value for this property.

Example

```
FTPClient.Target = "results"  
FTPClient.NewName = "results.txt"  
FTPClient.FileAction = FILE_ACTION_RENAME
```

Note:

This property is only used when using the FileAction and DirAction properties. For new applications set the NewName parameter in the RemoteFile or RenameDir methods.

2.23 Notify

Summary

Notify the number of bytes transferred in a direct (file-based) file transfer.

Description

The Notify property is used to specify whether the application should receive any OnTransfer events during a file transfer operation. If the Notify property is set to True, then one or more OnTransfer events will be fired. Otherwise, no OnTransfer events will occur.

The OnTransfer event occurs during a file based transfer of data from the local machine to a remote file or from the remote machine to a local file. The event contains the number of bytes of data transferred between the two machines.

As the file transfer proceeds, one or more OnTransfer events will occur to deliver the number of bytes transferred so far between the FTP server and the client. The application can process this data and display the status of the file transfer (for example, update a progress bar or text control).

This property can be changed at any time except during a file transfer operation. The default value of this property is False, which means that the application will not receive the OnTransfer event.

Example

```
FTPClient.Notify = True
```

2.24 Passive

Summary

Transfer file in passive mode.

Description

Some applications may want to request the FTP Server to listen for a data port and wait for the connection rather than initiate the process when a data transfer request comes in. The application can achieve this by setting the Passive property to True before initiating a file transfer request. To turn off passive mode file transfer, set the Passive property to False.

This property sets the LastResult property during run time and after a connection has been established. The value of the LastResult property can be checked to determine if any error occurred while trying to set the Passive property.

This property can be changed at any time. The default value of this property is False.

Example

```
FTPClient.Passive = True
```

2.25 Password

Summary

Specifies the Password used while connecting to the FTP server.

Description

The Password property is used during the login process to the server. Most FTP servers require both a user name and a correct password before a session can be established. Before connecting to a server, the application must also set the User property to the correct user name.

For security reasons, both the user name and the password are usually obtained from the user just before a connection is established. If security is not an important issue and the application will always connect to the same FTP server with the same user name and password, then these two properties can be set at design time.

For additional security, the FTP service may reside on a port other than the default port 21. If the application prompts the user for a user name and a password, then adding a field to allow the user to enter an FTP service port should be provided. Please check the reference page of the Port property for more details.

Usually for accounting purposes, some FTP servers require the user to enter an account name along with the user name and the password. If the application prompts the user for a user name and a password, then adding a field to allow the user to enter an account name should be provided. Please check the reference page of the Account property for more details.

This property can be changed at design time and at run time before a connection has been established. There is no default value for this property.

Example

```
FTPClient.Host = "speedy"  
FTPClient.User = "santa"  
FTPClient.Password = "north pole"  
FTPClient.Account = ""  
FTPClient.Action = ACTION_CONNECT
```

Note

When using the Connect or FwConnect methods the password is set through the password parameter.

2.26 Port

Summary

Specifies the FTP service port number on the FTP server.

Description

The Port property specifies the remote port on the FTP server on which the FTP service resides. Most FTP services listen for connection requests on port 21. Sometimes, usually for security reasons, port numbers other than 21 are used for FTP connections.

If the application will be connecting to FTP services on a different port, then the Port property must be set before the connection attempt is made. An application may even want to query the user for the correct port before connecting. The ActiveX control does not verify the setting of the Port property and any value is therefore legal.

If an application prompts the user for the user name and password before connecting to the server, then adding fields to allow the user to enter an optional account name (see the Account property reference page) and an FTP server port should be provided. If the login parameters are specified at design time, then the local system administrator should be consulted to determine if an account name or a port other than the default port are required.

This property can be changed at design time and at run time before a connection has been established. The default value for this property is port 21.

Example

```
FTPClient.Port = 21
Result = FTPClient.Connect ("speedy.distinct.com", "santa", "north pole", "")
If Result = False Then
    MsgBox "Unable to connect to server", 64, "Sample Program"
End If
```

2.27 ProxyHost

Summary

Name or dotted decimal Internet address of the proxy server to be used for the FTP connection.

Description

The ProxyHost property specifies the name or internet address of a proxy server. This property must be set before an FTP session can be established through a proxy server. Before connecting to a ftp server through a proxy server, the application must also set the ProxyUser, ProxyPassword, ProxyPort and ProxyType properties. There are three possible ways of specifying the address of a proxy server.

Machine Name

An application only needs to specify the name of the proxy server if the server is located on the same network as the local PC or if the internet address of the server is defined in the local host table. If the proxy server is not on the local network, then the underlying protocol will route the traffic through a gateway. If the proxy server is not defined in the local host table, then the underlying protocol will contact the domain server to resolve the internet address of the server.

Machine and Domain Name

An application needs to specify the machine name and the domain name if the proxy server is not located on the same network as the local PC. Fully domain qualified machine names are written from left to right in ascending order (for example, *genie.distinct.com*). If both machine and domain names are specified, then the underlying protocol will contact the domain server to resolve the internet address of the server.

Internet Address

Sometimes the user knows only the internet address of the proxy server that he or she wants to use. In this case, the internet address can be entered in the dotted decimal notation (for example, *127.43.101.12*). If the proxy server identified by this address is not on the local network, then the underlying protocol will route the traffic through a gateway.

This property can be changed at design time and at run time before a connection has been established. There is no default value for this property.

Example

```
FTPClient.ProxyHost = ftp.distinct.com
FTPClient.ProxyPort = 80
FTPClient.ProxyUser = "anonymous"
FTPClient.ProxyPassword = "distinct"
FTPClient.ProxyType = PROXY_TYPE_HTTP
Result = FTPClient.Connect ("speedy.distinct.com", "santa", "north pole", "")
```

2.28 ProxyPassword

Summary

Specifies the password used for HTTP proxy server if authentication is required.

Description

The ProxyPassword property is used during the login process to the ftp server when the connection is going through a proxy server. Some proxy servers require both a user name and a password for user authentication. Before connecting to a ftp server through a proxy server, the application must also set the ProxyUser if a username and password are required, ProxyHost, ProxyPort and ProxyType property.

For security reasons, both the user name and the password are usually obtained from the user just before a connection is established. If security is not an important issue and the application will always connect to a particular FTP server through the same proxy server with the same user name and password, then these two properties can be set at design time.

This property can be changed at design time and at run time before a connection is established. There is no default value for this property.

Example

```
FTPClient.ProxyHost = ftp.distinct.com
FTPClient.ProxyPort = 80
FTPClient.ProxyUser = "anonymous"
FTPClient.ProxyPassword = "distinct"
FTPClient.ProxyType = PROXY_TYPE_HTTP
Result = FTPClient.Connect ("speedy.distinct.com", "santa", "north pole", "sc001d")
```

2.29 ProxyPort

Summary

Specifies the port number on the HTTP proxy server through which an FTP connection will be made.

Description

The ProxyPort property specifies the remote port on the proxy server on which the proxy service resides.

The ProxyPort property must be set before the connection through the proxy server is made. The FTP ActiveX control does not verify the setting of the ProxyPort property. The user is therefore responsible for entering a valid port number.

This property can be changed at design time and at run time before a connection has been established. There is no default value for this property.

Example

```
FTPClient.ProxyHost = ftp.distinct.com
FTPClient.ProxyPort = 80
FTPClient.ProxyUser = "anonymous"
FTPClient.ProxyPassword = "distinct"
FTPClient.ProxyType = PROXY_TYPE_HTTP
Result = FTPClient.Connect ("speedy.distinct.com", "santa", "north pole", "sc001d")
```

2.30 ProxyType

Summary

Specifies the type of proxy server through which the FTP connection will be established.

Description

The ProxyType property specifies the proxy server type on which the proxy service resides. This release only supports HTTP proxy (SOCKS is also supported through a different set of properties)

The ProxyType property must be set before the connection through the proxy server is made. It should be set to one of the following legal values:

```
PROXY_TYPE_NONE  
PROXY_TYPE_HTTP
```

This property can be changed at design time and at run time before a connection has been established. Default value for this property is PROXY_TYPE_NONE.

Example

```
FTPClient.ProxyHost = ftp.distinct.com  
FTPClient.ProxyPort = 80  
FTPClient.ProxyUser = "anonymous"  
FTPClient.ProxyPassword = "distinct"  
FTPClient.ProxyType = PROXY_TYPE_HTTP  
Result = FTPClient.Connect ("speedy.distinct.com", "santa", "north pole", "sc001d")
```

2.31 ProxyUser

Summary

Specifies the user name required by the proxy server if the server requires user authentication.

Description

The ProxyUser property is used when the user tries to establish an FTP connection through a proxy server. Some proxy servers require that the user is authenticated by entering a user name and password. In this case ProxyPassword must also be set. The application must also set the ProxyHost, ProxyPort and ProxyType properties.

For security reasons, both the user name and the password are usually obtained from the user just before a connection is established. If security is not an important issue and the application will always connect to an FTP server through the same proxy server with the same user name and password, then these two properties can be set at design time.

This property can be set at design time or at run time before a connection has been established. There is no default value for this property.

Example

```
FTPClient.ProxyHost = ftp.distinct.com
FTPClient.ProxyPort = 80
FTPClient.ProxyUser = "abc"
FTPClient.ProxyPassword = "def"
FTPClient.ProxyType = PROXY_TYPE_HTTP
Result = FTPClient.Connect ("speedy.distinct.com", "santa", "north pole", "sc001d")
```

2.32 Quote

Summary

The Quote property invokes the FTP QUOTE command which allows a user to send a specific command to the FTP Server.

Description

The QUOTE property is used to send a command to the FTP server. The command is sent *as is* to the FTP server over the established connection. For example, setting the Quote property to **CDUP** is identical to calling the ParentDir method.

The Quote property cannot be used to obtain a directory listing or for transferring files, but it can be used for any other FTP command. For example, an application may set the Quote property to **PWD** to get the current working directory. The **PWD** command will be sent to the server and the response from the server will be delivered to the application with an OnReceive event. The result of this operation is identical to accessing the CurrentDir property.

The server's response to the FTP command can be a single line response (for example **PWD**) or a multiple line response (for example **HELP**).

The value of the LastResult property can be checked to determine if the FTP command was executed successfully.

The Quote property can be set at run time only. There is no default value for this property.

Example

```
FTPClient.Quote = "HELP"
```

Note

The QUOTE command is typically used to execute commands on the server that are not directly available from the ActiveX itself. The QUOTE command is essential to allow the user to access servers that require system-specific commands (e.g., SITE or ALLO), or to invoke new or optional features.

2.33 RemoteFile

Summary

Specifies the name of remote file to transfer.

Description

The RemoteFile property specifies the name of the remote file to be used during a file transfer. File transfers are initiated by setting the FileAction property to FILE_ACTION_GET or FILE_ACTION_PUT.

This property must be set to the name of a file in the current remote working directory for both file and event based file transfers.

This property can be changed at any time except during a file transfer. There is no default value for this property.

Example

```
FTPClient.RemoteFile = "test.exe"  
FTPClient.LocalFile = "c:\test.exe"  
FTPClient.FileAction = FILE_ACTION_GET
```

Note

For new applications specify the source or destination in the GetFile, PutFile, Remote and RemoteAppend methods instead of setting this property.

2.34 RemoteId

Summary

Remote connection id.

Description

The RemoteId property specifies the connection id of the FTP client to be used during a remote to remote file transfer.

The RemoteId property specifies the connection id of the second FTP client involved in the remote to remote transfer. Remote to remote file transfer needs two FTP clients. For example FTPClient1 and FTPClient2. To transfer a file between machine A and machine B: connect FTPClient1 to machine A and connect FTPClient2 to machine B. Then, set FTPClient1.RemoteId to FTPClient2.Id, and initiate the transfer by setting the FTPClient1.FileAction property to FILE_ACTION_REMOTE or FILE_ACTION_REMOTE_APPEND. This will transfer a file from machine A to machine B. There is no **local I/O** involved in the transfer.

To transfer a file from machine B to machine A, connect FTPClient1 to machine A, and connect FTPClient2 to machine B. Set FTPClient2.RemoteId to FTPClient1.Id and initiate the transfer by setting the FTPClient2.FileAction property to FILE_ACTION_REMOTE or FILE_ACTION_REMOTE_APPEND.

This property must be set to the connection id of the other FTP client involved in the remote to remote transfer, before the FileAction property is set to FILE_ACTION_REMOTE or FILE_ACTION_REMOTE_APPEND. This property can be changed at any time except during a file transfer. There is no default value for this property.

Example

```
FTPClient.RemoteId = FTPClient2.Id  
FTPClient.LocalFile = "test.exe"  
FTPClient.RemoteFile = "abc.exe"  
FTPClient.FileAction = FILE_ACTION_REMOTE
```

Note

This property only needs to be set if you are using the FileAction property. Otherwise use the RemoteId parameter in the RemoteAppend or Remote methods instead.

2.35 RetryCount

Summary

Specifies the number of times the file transfer will restart following a failed transfer.

Description

The `RetryCount` property is used during a file transfer using the `PutFileWithRestart` or `GetFileWithRestart` methods. During file transfer if there is an abnormal termination, the restart command may be sent to start the file transfer from where it terminated. `RetryCount` specifies the number of time that the restart command can be sent during a single file transfer.

This property can be changed at design time and at run time before or after the connection has been established. The default value for this property is 0.

Example

```
FTPClient.RetryCount = 1  
FTPClient.GetFileWithRestart ("abc.txt", "abc.txt", "")
```

See Also:

`GetFileWithRestart` and `PutFileWithRestart` methods.

2.36 SendData

Summary

The Send data buffer

Description

The SendData property is used to send data in the OnSend event. If UseProperty is set to True, the control will get the data to send from the SendData property, otherwise it will get the data from the Buffer parameter. The UseProperty property should usually be set to False, and the Buffer parameter should be used to send data. This property should only be used in environments like Visual J++ where the control is unable to get the new value of the Buffer parameter from the OnSend event. See the UseProperty property for more information on when to use this property..

This property should be set in the OnSend event. This property can be read at any time. There is no default value for this property.

Example

```
Sub FTPClient_OnSend (Buffer As String, Length As Integer)
    Dim Message As String

    If Len(Message) > 0 Then
        FTPClient.SendData = Message ' send message
        Message = ""
    Else
        FTPClient.SendData = Message ' indicate end of message
    End If
End Sub
```

2.37 Target

Summary

Specifies the file or subdirectory on which to perform the next file or directory action.

Description

The Target property is used to specify the file or subdirectory on which to perform the next file or directory action. This property always specifies a file or subdirectory located in the current working directory on the FTP server.

For file operations, the Target property must be set before assigning FILE_ACTION_DELETE or FILE_ACTION_RENAME to the FileAction property. The NewName property must also be set before renaming a remote file.

For directory operations, the Target property must be set before assigning DIR_ACTION_CHANGE, DIR_ACTION_CREATE, DIR_ACTION_DELETE or DIR_ACTION_RENAME to the DirAction property. The NewName property must also be set before renaming a remote subdirectory.

This property can be changed at any time. There is no default value for this property.

Example

```
FTPClient.Target = "company1"  
FTPClient.DirAction = DIR_ACTION_CHANGE
```

Note

This property only needs to be used if you are using the FileAction or DirAction properties. Otherwise use the Target parameter in the RenameFile or RenameDir methods.

2.38 TransferMode

Summary

Specifies whether the file transfer will be event or file based.

Description

The TransferMode property determines if a file transfer occurs directly between the specified remote file and a local disk file or if events will be generated which will allow the application to transfer data to and from storage other than the local disk, such as memory, through DDE or the clipboard. The value of this property can be set to either one of the following two constants.

Value	Meaning
TRANSFER_MODE_FILE	Direct transfer between local and remote files.
TRANSFER_MODE_EVENT	Event based transfer to and from remote files.

When the TransferMode property is set to TRANSFER_MODE_FILE, then the FTP Client ActiveX control takes care of all input and output (I/O) operations required to get a file from the server or to put a file onto the server. All the application needs to do is to set the LocalFile and RemoteFile properties so that they uniquely identify one local and one remote file before starting the file transfer. Once the FILE_ACTION_GET or FILE_ACTION_PUT value is assigned to the FileAction property, then the ActiveX control completes the file transfer without generating any events or involving the application in any other way.

When the TransferMode property is set to TRANSFER_MODE_EVENT, then a file is transferred through events. This means that one or more OnSend events (in case of a put or append operation) or one or more OnReceive events (in case of a get operation) will be fired. An OnSend event asks the application to provide up to a maximum number of bytes of data, which are to be saved in the remote file. OnSend events are generated until the application indicates that there is no more data to send. An OnReceive event delivers a portion of the data from the remote file to the application. Event based file transfer allows an application to move data to and from storage other than the local disk, such as through DDE or the clipboard. For event based file transfer, the LocalFile property is ignored and can be left blank.

The FileMode and EventMode methods can also be used to set the transfer mode. Please check the reference pages of these methods for more detailed information on their usage.

This property can be changed at any time except during a file transfer. The default value for this property is TRANSFER_MODE_FILE.

Example

```
FTPClient.RemoteFile = "test.exe"
FTPClient.LocalFile = "c:\test.exe"
FTPClient.TransferMode = TRANSFER_MODE_FILE
FTPClient.TransferType = TRANSFER_TYPE_BINARY
FTPClient.FileAction = FILE_ACTION_GET
```

Note

For new applications use the EventMode or FileMode methods to set the file transfer mode.

2.39 TransferType

Summary

Specifies whether the file transfer will take place in ASCII or binary mode.

Description

The TransferType property determines if any type of file conversion will be performed by the server during file transfers. The value of this property can be set to either one of the following two constants.

Value	Meaning
TRANSFER_TYPE_ASCII	Convert CR and LF to LF and vice versa.
TRANSFER_TYPE_BINARY	Transfer files without conversion.

When the TransferType property is set to TRANSFER_TYPE_BINARY, then no file conversion is performed. Files are sent and received exactly as they are saved on the originating machine. This allows undisturbed transfers of non-text files, such as compressed files, archives and executables.

When the TransferType property is set to TRANSFER_TYPE_ASCII, then carriage return and line feed pairs are converted to line feeds during put or append operations and line feeds are expanded into carriage return and line feed pairs during get operations. The ASCII mode conversion is normally used to convert text files to a format suitable for text editors on the destination machine.

The value of the LastResult property can be checked to determine if the action has been performed successfully. Setting the TransferType property before a session is established will always be successful.

The Ascii and Binary methods can also be used to set the transfer type. Please check the reference pages of these methods for more detailed information on their usage.

This property can be changed at any time except during a file transfer. The default value for this property is TRANSFER_TYPE_ASCII.

Example

```
FTPClient.RemoteFile = "test.exe"  
FTPClient.LocalFile = "c:\test.exe"  
FTPClient.TransferMode = TRANSFER_MODE_FILE  
FTPClient.TransferType = TRANSFER_TYPE_BINARY  
FTPClient.FileAction = FILE_ACTION_GET
```

Note

For new applications use the Ascii or Binary methods to set the data transfer type.

2.40 TransmissionMode

Summary

Get the current Transmission mode.

Description

The TransmissionMode property is used to get the current transmission mode. There are two legal values for this property:

TRANSMISSION_MODE_STREAM
TRANSMISSION_MODE_BLOCK

You can also use the BlockMode methods to set the transmission mode to block mode and StreamMode function to set it to stream mode.

The default value for this property is TRANSMISSION_MODE_STREAM.

Example

Mode = FTPClient.**TransmissionMode**

Note

For new applications use the BlockMode or StreamMode methods to set the transmission mode.

2.41 UseProperty

Summary

Determines whether the Buffer parameter or the SendData property will be assigned data in the OnSend event.

Description

The UseProperty property is used to specify whether the Buffer parameter or the SendData property should be assigned data in the OnSend event. If UseProperty is set to True, the control will get the data to send from the SendData property, otherwise it will get the data from the Buffer parameter.

The UseProperty property is generally set to False, and the event parameter should be used in most cases. It should only be set to True in environments like Visual J++ where the control is unable to get the new value of the event parameter. This is because Visual J++ has to use VBScript to catch and pass the ActiveX events, and VBScript is unable to pass back parameters to the control.

This property should be set before setting the FileAction property to FILE_ACTION_PUT (or calling the PutFile method) in an event based file copy. This property can be read at any time. The default value for this property is False.

Example

```
FTPClient.UseProperty = False
```

2.42 User

Summary

User name used while connecting to FTP server.

Description

The User property is used during the login process to the server. Most FTP servers require both a user name and a correct password before a session can be established. Before connecting to a server, the application must also set the Password property to the correct password.

For security reasons, both the user name and the password are usually obtained from the user just before a connection is established. If security is not an important issue and the application will always connect to the same FTP server with the same user name and password, then these two properties can be set at design time.

For additional security, the FTP service may reside on a port other than the default port 21. If the application prompts the user for a user name and a password, then adding a field to allow the user to enter an FTP service port should be provided. Please check the reference page of the Port property for more details.

Usually for accounting purposes, some FTP servers require the user to enter an account name along with the user name and the password. If the application prompts the user for a user name and a password, then adding a field to allow the user to enter an account name should be provided. Please check the reference page of the Account property for more details.

This property can be changed at design time and at run time before a connection has been established. There is no default value for this property.

Example

```
FTPClient.Host = "speedy.distinct.com"  
FTPClient.User = "santa"  
FTPClient.Password = "north pole"  
FTPClient.Account = ""  
FTPClient.Action = ACTION_CONNECT
```

Note

For new applications use the User parameter in the Connect or FwConnect methods to specify the user name when connecting to a FTP server.

2.43 UseVariant

Summary

Send and receive binary or ascii data.

Description

The UseVariant property is used to specify whether data is to be received or sent in binary or ASCII form. If this property is set to True, the OnReceiveB event will be fired when data arrives; otherwise, the OnReceive event is fired. Similarly, the OnSendB event will be fired when there is data to be sent; otherwise, the OnSend event is fired.

This property should be set before any data arrives or before sending any data. This property can be read at any time. The default value for this property is False.

Example

```
FTPClient.UseVariant = True
```

Note

This is used only during an event based file transfer.

2.44 Wildcards

Summary

Directory wildcards used for directory listing.

Description

The Wildcards property is used to limit the directory entries reported during a directory list operation to just those files and subdirectories matching the given wildcard expression. Directory listings are initiated by setting the DirAction property to DIR_ACTION_LIST.

Only one wildcard expression, such as "*.TXT", can be specified. It is not possible to specify composite wildcards, such as "*.EXE *.COM" or "*.* - *.TMP".

If the Wildcards property is left blank, then all files and subdirectories will be listed. This is identical to setting this property to "*.*".

This property can be changed at any time except during a directory list operation. There is no default value for this property.

Example

```
FTPClient.Wildcards = "*.c"  
FTPClient.ListType = LIST_TYPE_SHORT  
FTPClient.DirAction = DIR_ACTION_LIST
```

Note

For new applications use the wildcards parameter in the ListDir method.

3 FTP Client Events

3.1 OnClose

Summary

FTP connection has been closed.

Description

The OnClose event occurs usually in response to the Disconnect or Abort method (or in response to setting the Action property to ACTION_DISCONNECT or ACTION_ABORT). In some cases, the FTP server may close a connection (for example because of a long period of inactivity). This will also trigger an OnClose event. In this case, the application must still set the Action property to ACTION_DISCONNECT (or call the Disconnect method) to free up all the resources allocated for the connection. However, this should not be done during the OnClose event because it might result in an infinite loop.

If this event occurs in response to setting the Action property to ACTION_DISCONNECT (or in response to the Disconnect method), it will occur before the statement following the assignment of ACTION_DISCONNECT to the Action property (or the call to the Disconnect method) is reached.

Normally, an application should simply set a flag in response to this event. Then, this flag can be checked directly after the ACTION_DISCONNECT (or the Disconnect method) action to make sure that the connection was actually terminated.

Example

```
Sub FTPClient_OnClose ()  
    Connected = False  
End Sub
```

See Also

Disconnect and Abort methods

3.2 OnConnect

Summary

Notifies that the FTP connection has been established.

Description

The OnConnect event occurs in response to the Connect method (or in response to setting the Action property to ACTION_CONNECT). This event occurs immediately after the Connect method is called (or the Action property is set to ACTION_CONNECT) and before the next line of code immediately following is executed.

Normally, an application should simply set a flag in response to this event. Then, this flag can be checked directly after the ACTION_CONNECT action (or the Connect method) to make sure that the connection was actually established.

If the connection could not be established, then the OnError event will be called instead of the OnConnect event.

While handling the OnConnect event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box.

Example

```
Sub FTPClient_OnConnect ()
    Connected = False
    FTPClient.Connect ("speedy.distinct.com", "santa", "north pole", "")
    If Connect = False Then
        MsgBox "Unable to connect to server", 64, "Sample Program"
    End If
End Sub
```

Note

A file transfer cannot be initiated in this event.

3.3 OnError

Summary

Notifies that a local error has occurred.

Description

The OnError event occurs when a property is accessed in an illegal way or when a connection with the FTP server cannot be established. The following describes all possible error codes delivered by this event.

Value	Meaning
ERR_CANNOT_CHANGE_XFER_TYPE	Cannot change transfer type during file transfer. This may have been caused by calling the GetFile, PutFile or AppendFile methods during a file transfer.
ERR_CANNOT_CHANGE_XFER_MODE	Cannot change transfer mode during file transfer. This may have been caused by calling the GetFile, PutFile or AppendFile methods during a file transfer.
ERR_CANNOT_CHANGE_LIST_TYPE	Cannot change list type during directory listing. This may have been caused by calling the List method while a directory listing was in progress.
ERR_CANNOT_CHANGE_PORT ERR_PORT_UNDEFINED	Cannot change the remote port while connected. Remote port must be specified before connecting. The Port property must be set before calling the Connect method.
ERR_HOST_UNDEFINED	Remote host must be defined before connecting.
ERR_IN_TRANSFER	A file transfer or directory listing is in progress.
ERR_CANNOT_CONNECT	Unable to connect. The port or the host property may be set incorrectly or the host may be down.
ERR_NEED_ACCOUNT	The server requires account information to complete login.
ERR_CANNOT_LOG_IN	Unable to login. The user name or password may be incorrect.
ERR_NOT_CONNECTED	Must be connected to perform this operation.
ERR_IN_ACTION	Another action is in progress.
ERR_NO_DIR_TARGET	Target directory is not specified.
ERR_NO_NEW_DIR_NAME	New directory name is not specified.
ERR_NO_FILE_TARGET	Target file not specified.
ERR_NO_NEW_FILE_NAME	New file name not specified.
ERR_NO_REMOTE_FILE	Remote file not specified.
ERR_NO_LOCAL_FILE	Local file not specified.
ERR_CANNOT_OPEN_LOCAL_FILE	Unable to open local file.
ERR_FW_SERVER_NOT_DEFINED	Firewall server is not specified. Connections to a remote host via a firewall can only be established if the firewall server is defined.

Example

```
Sub FTPClient_OnError (ErrorCode As Integer)
    If ErrorCode = ERR_CANNOT_CONNECT Then
        MsgBox "Unable to connect to remote host", 64, "Sample Program"
    End If
End Sub
```

3.4 OnList

Summary

An OnList event is fired for each item in the directory listing received.

Description

The OnList event occurs once for every matching file or subdirectory, in your current remote directory, specified by the ListDir Method. A directory listing operation is triggered by calling the ListDir method (or by setting the DirAction property to DIR_ACTION_LIST). OnList events will occur directly after the ListDir method is called (or DirAction property is set) and before the line of code immediately following is executed.

The Wildcards property can be used to restrict the directory listing to only those files and subdirectories which match a given wildcard.

The format of each directory listing line is controlled by setting the ListType property either to LIST_TYPE_SHORT (or by calling the ShortList method) for short listings or to LIST_TYPE_LONG (or by calling the LongList method) for long listings. OnList events generated for a short listing will only include the name of a subdirectory or of a file, but long listing lines may include any number of additional pieces of information. Most servers report the size of a file, the time and date of the last modification and the file attributes in a long listing.

The format of long directory listing entries varies from one FTP server to another. Fields, such as name, size, date and time, may be displayed differently and may not be in the same order. It is up to the application to interpret the incoming lines. In most cases, the directory entry lines can simply be displayed to the user without any further processing. The format can be parsed using the Parse() method.

While handling the OnList event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box. A substantial delay could cause the underlying protocol to time out and terminate the listing prematurely.

Example

```
Sub FTPClient_OnList (Entry As String)
    Listing.AddItem Entry
End Sub
```

3.5 OnReceive

Summary

Notifies that more data, during an event based transfer, has been received.

Description

The OnReceive event occurs during an event based transfer of a remote file to the local machine. An event driven get operation is started by calling the GetFile method (or by setting the FileAction property to FILE_ACTION_GET) and calling the EventMode method (or with the TransferMode property set to TRANSFER_MODE_EVENT). The remote file must be specified with the RemoteFile property, but the LocalFile property is ignored and can be left blank.

Instead of writing incoming data from the remote file directly to a local file, an event based get operation delivers the data in segments to the application with one or more OnReceive events. The event passes data to the application in a string buffer. It is up to the application to process the data. The buffer could, for example, be written to a disk file, sent to another application using DDE, copied into the clipboard or scanned for the desired information and then discarded.

If the application does not want to receive any more OnReceive events, then the transfer can be aborted by setting the FileAction property to FILE_ACTION_ABORT (or by calling the AbortFile method) during the OnReceive event.

While handling the OnReceive event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box. A substantial delay could cause the application to not receive subsequent OnReceive events.

Example

```
Sub FTPGet ()
    FTPClient.UseVariant = False
    FTPClient.EventMode()
    FTPClient.GetFile ("test.exe", "")
End Sub
' OnReceive event fired to deliver data

Sub FTPClient_OnReceive (Buffer As String)
    Listing.AddItem Buffer
End Sub
```

Note

The EventMode() method must be called before calling the GetFile() method.

See Also

OnReceiveB

3.6 OnReceiveB

Summary

Notifies that more data, during event based transfer, has been received as binary.

Description

The OnReceiveB event occurs during an event based transfer of a remote file to the local machine. An event driven get operation is started by calling the GetFile method (or by setting the FileAction property to FILE_ACTION_GET) following a call to the EventMode method (or with the TransferMode property set to TRANSFER_MODE_EVENT). The remote file must be specified with the RemoteFile property, but the LocalFile property is ignored and can be left blank. The UseVariant property must be set to True. If the UseVariant property is False, then the OnReceive event will be received instead.

Instead of writing incoming data from the remote file directly to a local file, an event based get operation notifies the application with one or more OnReceiveB events. The event notifies the application how many bytes of data are ready to be retrieved. It is up to the application to call the ReceiveB method to retrieve and process the data. The buffer could, for example, be written to a disk file or sent to another application using DDE.

If the application does not want to receive any more OnReceiveB events, then the transfer can be aborted by setting the FileAction property to FILE_ACTION_ABORT (or by calling the AbortFile method) during the OnReceiveB event.

While handling the OnReceiveB event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box. A substantial delay could cause the application to not receive subsequent OnReceiveB events.

Example

```
Sub FTPGet ()
    FTPClient.UseVariant = True
    FTPClient.EventMode()
    FTPClient.GetFile ("test.exe", "")
End Sub
' OnReceiveB event fired to deliver data

Sub FTPClient_OnReceiveB (Bytes As Long)
    Dim Buffer() As Byte
    Dim Siz As Long
    Dim i As Integer

    Siz = FTPClient.ReceiveB (Buffer, Bytes)
    If Siz > 0 Then
        Open "c:\abc.exe" For Binary Access Write As #1
        For i = 1 To Siz
            Put #1, i, Buffer (i)
        Next i
        Close #1
    Else
        MsgBox "Cannot receive binary data", 64, "Sample Program"
    End If
End Sub
```

3.7 OnReceiveMarker

Summary

Marker data during event based transfer has been received.

Description

The OnReceiveMarker event occurs during an event based transfer of a file in block mode. OnReceiveMarker event is interleaved with OnReceive events according to the marker frequency. An event driven get operation is started by calling the GetFileWithRestart method (or by setting the FileAction property to FILE_ACTION_GET) with the call to the EventMode method (or with the TransferMode property set to TRANSFER_MODE_EVENT). An event driven put operation is started by calling the PutFileWithRestart method (or by setting the FileAction property to FILE_ACTION_PUT with the call to the EventMode method or with the TransferMode property set to TRANSFER_MODE_EVENT). The remote file must be specified with the RemoteFile property, but the LocalFile property is ignored and can be left blank in case of transferring the file from server to local machine.

Instead of writing incoming marker data from server directly to a local marker file, an event based operation in block mode delivers the marker to the application with one or more OnReceiveMarker events. The event passes marker data to the application in a string buffer. It is up to the application to process the marker data. The buffer could, for example, be written to a disk file. In case of an abnormal termination of file transfer, user can use GetFileWithRestart or PutFileWithRestart function with any of the markers previously received. This will start the transfer of file from the specified marker position. It is user's responsibility to position the local file according to marker. This way user does not need to transfer the whole file again.

In the case of a Get, each marker should be stored together with the corresponding byte position of the block of data received in the local file immediately preceding the marker. If the file transfer is terminated abnormally, instead of transferring the whole file again, user can position the local file to the most recent position stored and send a GetFileWithRestart command with the corresponding stored marker.

In case of a Put operation, OnReceiveMarker delivers the local file position as well as server marker for that position. User should store this buffer in a file. The format of the buffer received is h-l=m, where h is the high word of local file position, l is the low word of local file position and m is the server marker for the local file position. In case of abnormal file transfer termination, user may position the local file to the most recent file position and send PutFileWithRestart function with the corresponding server marker.

If the application does not want to receive any more OnReceiveMarker events, then the transfer can be aborted by setting the FileAction property to FILE_ACTION_ABORT (or by calling the AbortFile method) during the OnReceiveMarker event.

While handling the OnReceiveMarker event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box. A substantial delay could cause the application to not receive subsequent OnReceive and OnReceiveMarker events.

Example

```
Sub FTPClient_OnReceiveMarker (Buffer As String)
" The buffer parameter contains the marker in the form of a numeric string e.g. "24968"
    Listing.AddItem Buffer
End Sub
```

3.8 OnReceiveMarkerB

Summary

Marker data during event based transfer has been received as binary.

Description

The OnReceiveMarkerB event occurs during an event based transfer of a file in block mode. OnReceiveMarkerB events are interleaved with OnReceiveB events depending upon the marker frequency. To do an event based Get call the GetFileWithRestart method with the Localfile parameter set to an empty string. To do an event based Put call the PutFileWithRestart method with the LocalFile parameter set to an empty string. This means that you must call the BolckMode method and the EventMode method and set the Use Variant property to TRUE before initiating the file transfer.

Instead of writing incoming marker data from server directly to a local marker file, an event-based operation notifies the application with one or more OnReceiveMarkerB events. The event notifies the application how many bytes of marker data are ready to be retrieved. It is up to the application to call the ReceiveB method to retrieve and process the marker data. The buffer could, for example, be written to a disk file, and used in abnormal termination of file transfer. Function GetFileWithRestart or PutFileWithRestart could be used with marker to transfer the file from the termination point.

In the case of a Get, each marker should be stored together with the corresponding byte position of the block of data received in the local file immediately preceding the marker. If the file transfer is terminated abnormally, instead of transferring the whole file again, user can position the local file to the most recent position stored and send a GetFileWithRestart command with the corresponding stored marker.

In case of a Put operation, OnReceiveMarker delivers the local file position as well as server marker for that position. User should store this buffer in a file. The format of the buffer received is h-l=m, where h is the high word of local file position, l is the low word of local file position and m is the server marker for the local file position. In case of abnormal file transfer termination, user may position the local file to the most recent file position and send PutFileWithRestart function with the corresponding server marker.

If the application does not want to receive any more OnReceiveMarkerB events, then the transfer can be aborted by setting the FileAction property to FILE_ACTION_ABORT (or by calling the AbortFile method) during the OnReceiveMarkerB event.

While handling the OnReceiveMarkerB event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box. A substantial delay could cause the application to not receive subsequent OnReceiveMarkerB and OnReceiveB events.

Example

```
Sub FTPClient_OnReceiveMarkerB (Bytes As Long)
    Dim Buffer() As Byte
    Dim Siz As Long
    Dim i As Integer

    Siz = FTPClient.ReceiveB (Buffer, Bytes)
    If Siz > 0 Then
        Open "c:\abc.mar" For Binary Access Write As #1
        For i = 1 To Siz
            Put #1, i, Buffer (i)
        
```

```
        Next i
        Close #1
    Else
        MsgBox "Cannot receive binary marker data", 64, "Sample Program"
    End If
End Sub
```

3.9 OnSend

Summary

The OnSend event is fired during an event based put or append operation to get the next block of data.

Description

The OnSend event occurs during an event based transfer of data from the local machine to a remote file. An event driven put or append operation is started by calling the PutFile or AppendFile method (or by setting the FileAction property to FILE_ACTION_PUT or FILE_ACTION_APPEND) with the call to the EventMode method (or the TransferMode property set to TRANSFER_MODE_EVENT). The remote file must be specified with the RemoteFile parameter, but the LocalFile parameter is ignored and can be left blank.

With each event, the application is given a string buffer and an integer specifying the maximum number of bytes that can be copied into the string buffer. It is up to the application to obtain the data. The data copied into this buffer could, for example, be read from a disk file, received from another application using DDE or copied from the clipboard. Although less than the requested number of bytes can be assigned to the string buffer, it is generally preferable for performance reasons to copy the maximum number of bytes if they are available.

In some environments, such as Visual J++, the new value assigned to the *Buffer* parameter can not be successfully retrieved by the control. In these cases set UseProperty to True, and assign data to the SendData property instead of the *Buffer* parameter. The UseProperty property must be set to True before the FileAction property is set to FILE_ACTION_PUT or FILE_ACTION_APPEND (or before calling the PutFile or AppendFile method) In this case treat the SendData property exactly as you would the *Buffer* parameter. If UseProperty is True the *Length* parameter will be ignored.

To indicate that no more data is available to send, the application should clear the string buffer by assigning an empty string ("") to it. After returning an empty string buffer, no more OnSend events will be generated. The application can also terminate an event based transfer by setting the FileAction property to FILE_ACTION_ABORT (or by calling the AbortFile method can also be used) during the OnSend event. In this case, the returned string buffer is ignored by the control.

While handling the OnSend event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box.

Example

```
Sub FTPClient_OnSend (Buffer As String, Length As Integer)
    Dim Message As String

    If Len(Message) > 0 Then
        Buffer = Message           ' send message
        Message = ""
    Else
        Buffer = Message         ' indicate end of message
    End If
End Sub
Sub FTPSend
    FTPClient.Eventmode()
    FTPClient.PutFile ("test.txt")
' OnSend event fired to get data
EndSub
```

3.10 OnSendB

Summary

The OnSendB is fired during an event-based put or append to get the next block of binary data.

Description

The OnSendB event occurs during an event based transfer of data from the local machine to a remote file. An event driven put or append operation is started by calling the PutFile or AppendFile method (or by setting the FileAction property to FILE_ACTION_PUT or FILE_ACTION_APPEND) with the call to the EventMode method (or with the TransferMode property set to TRANSFER_MODE_EVENT). The remote file must be specified with the RemoteFile parameter, but the LocalFile parameter is ignored and can be left blank. The UseVariant property must be set to True to receive this event. If the UseVariant property is False, then the OnSend event will be received instead. The Byte argument passed to this event indicates the maximum number of bytes of data that can be sent at that moment.

With each event, the application must call the SendB method and pass the data and the size. The data could, for example, be read from a disk file, received from another application using DDE or copied from the clipboard.

To indicate that no more data is available to send, the application should pass 0 as the number of bytes to send to the SendB method. After passing 0 bytes to the SendB method, no more OnSendB events will be generated. The application can also terminate an event based transfer by setting the FileAction property to FILE_ACTION_ABORT (or by calling the AbortFile method can also be used) during the OnSendB event. In this case, the returned string buffer is ignored by the control.

While handling the OnSendB event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box.

Example

```
Sub FTPClient_OnSendB (Bytes As Long)
    Dim Buffer(1 to Bytes) As Byte
    Dim i As Integer

    If (Siz > Bytes) Then
        For i = 1 to Bytes
            Get #1, , Buffer(i)
        Next i
        Result = FTPClient.SendB(Buffer, Bytes)           ' send message
        Siz = Siz - Bytes
    ElseIf (Siz > 0) Then
        For i = 1 to Siz
            Get #1, , Buffer(i)
        Next i
        Result = FTPClient.SendB(Buffer, Siz)           ' send message
        Siz = 0
    Else
        Result = FTPClient.SendB (Buffer, 0)           ' end of message
    End If
End Sub
```

3.11 OnTransfer

Summary

The OnTransfer event fires the number of bytes transferred during file based transfer.

Description

The OnTransfer event occurs during a file based transfer of data from the local machine to a remote file or from the remote machine to a local file. The event contains the number of bytes of data transferred between the two machines.

A file driven get, put, or append operation can be initiated by setting the FileAction property to FILE_ACTION_GET, FILE_ACTION_PUT, or FILE_ACTION_APPEND (or by calling the GetFile, PutFile or AppendFile method) with the TransferMode property set to TRANSFER_MODE_FILE (or with the call to the FileMode method). The local file must be specified with the LocalFile property and the remote file must be specified with the RemoteFile property. The TransferType property must be set to TRANSFER_TYPE_ASCII or TRANSFER_TYPE_BINARY (or the Ascii or Binary method must be called).

This event will be fired only if the Notify property is set to True. If the Notify property is set to False, then no OnTransfer events will occur.

As the file transfer proceeds, one or more OnTransfer events will occur to deliver the number of bytes transferred so far between the FTP server and the client. The application can process this data and display the status of the file transfer (for example, update a progress bar or text control).

While handling the OnTransfer event, an application should not perform tasks, which have the potential of requiring a lot of time to complete, such as generating a message box.

Example

```
Sub FTPClient_OnTransfer (BytesCopied As Long)
    Label.Caption = Str$(BytesCopied) & " transferred"
End Sub
```

See Also

The Notify property

4 FTP Client Methods

4.1 Abort

Summary

Abort an FTP session.

Syntax

Boolean Abort ()

Description

The Abort method aborts a session to the FTP server. This method resets and closes the connection without properly closing down and should not be called under normal circumstances.

The Abort method takes no parameters and returns a boolean. If the connection is successfully reset and closed, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the Action property to ACTION_ABORT.

Example

```
Result = FTPClient.Abort ()
If Result = False Then
    MsgBox "Unable to abort", 64, "Sample Program"
Exit Sub
End If
```

4.2 AbortDir

Summary

Abort directory listing in progress.

Syntax

Boolean AbortDir ()

Description

The AbortDir method aborts a directory listing in progress. This method stops the application from receiving any more entries in the OnList event, which was invoked by the ListDir method or by setting the DirAction property to DIR_ACTION_LIST.

The AbortDir method takes no parameters and returns a boolean. If the directory listing in progress is successfully aborted, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the DirAction property to DIR_ACTION_ABORT.

Example

```
Result = FTPClient.AbortDir ()  
If Result = False Then  
    MsgBox "Cannot abort directory listing", 64, "Sample Program"  
End If
```

4.3 AbortFile

Summary

Abort file transfer in progress.

Syntax

Boolean AbortFile ()

Description

The AbortFile method aborts a file transfer in progress. This method aborts any get, put, or append operation in progress. This is usually done during an OnSend or OnReceive event to make sure that no further such events will occur.

The AbortFile method takes no parameters and returns a boolean. If the get, put, or append operation in progress is successfully aborted, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the FileAction property to FILE_ACTION_ABORT.

Example

```
Result = FTPClient.AbortFile ()  
If Result = False Then  
    MsgBox "Unable to abort file transfer", 64, "Sample Program"  
End If
```

4.4 Allocate

Summary

Allocate space on server.

Syntax

Boolean Allocate (*Bytes*)
Bytes Integer

Description

The function allocates the space specified by *Bytes* on the FTP server. If the allocation is successfully done, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

This method is typically called prior to a Put operation in order to reserve the space required to complete the file transfer.

Example

```
Result = FTPClient.Allocate (2048)
If Result = False Then
    MsgBox "Unable to allocate space", 64, "Sample Program"
End If
```

4.5 AppendFile

Summary

Append to a file on server.

Syntax

Boolean AppendFile (*LocalFile*, *RemoteFile*)

<i>LocalFile</i>	String
<i>RemoteFile</i>	String

Description

Files can be transferred to and from the FTP server either directly or through events depending on the setting of the TransferMode property or the call to the EventMode or FileMode method.

The AppendFile method takes a local file name (*LocalFile*) and a remote file name (*RemoteFile*) as its parameters and returns a boolean. For an event based file transfer, pass an empty string as the local file parameter. If the file was successfully transferred, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

In direct mode, a call to the AppendFile method will append the contents of the local file to the remote file. The local file and the remote file are passed as parameters to this method. This feature allows an application to transfer complete files of any size without doing any I/O itself. The local file parameter passed must contain a fully qualified path name since there is no current local directory.

When a file is transferred through events, then one or more OnSend events will be fired. An OnSend event asks the application to provide up to a maximum number of bytes of data, which are to be saved in the remote file. OnSend events are generated until the application indicates that there is no more data to send. Event based file transfer allows an application to move data to and from storage other than the local disk, such as through DDE or the clipboard. For event based file transfer, the local file parameter is ignored and can be left blank.

Calling this method is equivalent to setting the FileAction property to FILE_ACTION_APPEND.

Example

```
Result = FTPClient.AppendFile ("c:\test.exe", "test.exe")
If Result = False Then
    MsgBox "Cannot append to file on server", 64, "Sample Program"
End If
```

4.6 Ascii

Summary

Set transfer type to ASCII file transfer.

Syntax

Boolean Ascii ()

Description

The Ascii method sets the file transfer type to ASCII. That is, the server converts carriage return (CR) and line feed (LF) pairs to line feeds during put or append operations and line feeds are expanded into carriage return and line feed pairs during get operations. The ASCII mode conversion is normally used to convert text files to a format suitable for text editors on the destination machine.

The Ascii method takes no parameters and returns a boolean. If the transfer type can be successfully set to ASCII type file transfer, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the TransferType property to TRANSFER_TYPE_ASCII.

This method can be called at any time except during a file transfer.

Example

```
Result = FTPClient.Ascii ()
```

4.7 Binary

Summary

Set transfer type to binary file transfer.

Syntax

Boolean Binary ()

Description

The Binary method sets the file transfer type to binary. During file transfer, no file conversion is performed. Files are sent and received exactly as they are saved on the originating machine. This allows undisturbed transfers of non-text files, such as compressed files, archives and executables.

The Binary method takes no parameters and returns a boolean. If the transfer type can be successfully set to binary type file transfer, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the TransferType property to TRANSFER_TYPE_BINARY.

This method can be called at any time except during a file transfer.

Example

```
Result = FTPClient.Binary ()
```

4.8 BlockMode

Summary

Set transmission mode to block mode.

Syntax

Boolean BlockMode ()

Description

The method changes the transmission mode to block mode. Before this function is called the user needs to set the BlockSize property. Default value of the BlockSize property is 4096.

When a connection is opened, it defaults to STREAM mode. If you wish to initiate a transfer in block mode you must call this method prior to calling the PutFileWithRestart or the GetFileWithRestart methods.

This method can be called at any time except during a file transfer.

Example

```
Result = FTPClient.BlockMode ()
```

Note

Note that the majority of FTP servers that support the Restart following a failed transfer option ONLY support STREAM mode. This means if you make use of this feature, your application must be able to handle the situations where block mode is not supported.

4.9 Cancel

Summary

Abort any command or action that is in progress

Syntax

Boolean Cancel ()

Description

The Cancel method cancels any file transfer command or action that is in progress. Actions that can be aborted include trying to establish an FTP session, listing a directory, and any directory or file manipulations. After calling this method, the application should wait for the canceled command or action to return before attempting anything else with the connection.

The Cancel method takes no parameters and returns a boolean. If a command or action is successfully canceled, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the Action property to ACTION_CANCEL.

Example

```
Result = FTPClient.Cancel ()  
If Result = False Then  
    MsgBox "Cannot cancel current command or action", 64, "Sample Program"  
End If
```

4.10 ChangeDir

Summary

Change your current working directory on the FTP server.

Syntax

Boolean ChangeDir (*Target*)
Target String

Description

The ChangeDir method changes the current working directory on the FTP server to a specified target directory. This target directory can be a subdirectory of the current directory, ".." (identical to ParentDir) or a fully qualified path to a new working directory. This allows an application to change to any directory on the server in just one step.

The ChangeDir method takes a *Target* parameter and returns a boolean. If the current working directory can be successfully changed to the directory specified by the *Target* parameter, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the DirAction property to DIR_ACTION_CHANGE.

Example

```
Result = FTPClient.ChangeDir ("abc")
If Result = False Then
    MsgBox "Cannot change directory", 64, "Sample Program"
End If
```

4.11 Connect

Summary

Connect to FTP server.

Syntax

Boolean Connect (*Host*, *User*, *Password*, *Account*)

<i>Host</i>	String
<i>User</i>	String
<i>Password</i>	String
<i>Account</i>	String (note that this is not used by most FTP servers)

Description

The Connect method establishes a connection to the FTP server.

The Connect method takes a host name (*Host*), a user name (*User*), a password (*Password*) and an account (*Account*) as its parameters and returns a boolean. *Host* must be the name or internet address (in dotted decimal notation) of the FTP server. *User* and *Password* must contain a valid user name and password to complete the login process. Some servers also require that *Account* contain the name of an account for the user. If the server does not require the name of an account for the user, then *Account* can be set to an empty string.

If a connection is successfully established, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

If the connection can be established, then the OnConnect event will be fired. If the connection cannot be established, then the OnError event will be fired. These events will occur before the next statement (i.e. the statement following the call to the Connect method) is executed. The application should set a flag in the OnConnect and OnError events, so that it can determine if the session has been established or not. In addition, the application may want to display an error message in the OnError event to inform the user that the connection has not been established. Please check the reference page of the OnError event for a complete listing of error codes.

Calling this method is equivalent to setting the Action property to ACTION_CONNECT.

Example

```
Result = FTPClient.Connect ("speedy.distinct.com", "santa", "north pole", "sc001d")
If Result = False Then
    MsgBox "Unable to connect to server", 64, "Sample Program"
End If
```

Note

File transfer must NOT be initiated until this method has returned TRUE.

4.12 CreateDir

Summary

Create directory on server.

Syntax

Boolean CreateDir (*Target*)
Target String

Description

The CreateDir method creates a subdirectory on the FTP server with the specified target subdirectory name. This target subdirectory name cannot be the same as any existing subdirectory or file name.

The CreateDir method takes a *Target* parameter and returns a boolean. If the target subdirectory can be successfully created, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the DirAction property to DIR_ACTION_CREATE.

Example

```
Result = FTPClient.CreateDir ("test")
If Result = False Then
    MsgBox "Cannot create the specified subdirectory", 64, "Sample Program"
End If
```

4.13 DeleteFile

Summary

Delete file in current directory on server.

Syntax

Boolean DeleteFile (*Target*)
Target String

Description

The DeleteFile method deletes the file in the current directory of the FTP server with the specified target file name.

The DeleteFile method takes a *Target* parameter and returns a boolean. If the target file can be successfully deleted, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the FileAction property to FILE_ACTION_DELETE.

Example

```
Result = FTPClient.DeleteFile ("test.exe")
If Result = False Then
    MsgBox "Cannot delete file on server", 64, "Sample Program"
End If
```

4.14 Disconnect

Summary

Closes connection to the FTP server.

Syntax

Boolean Disconnect ()

Description

Once a connection is no longer needed, the session can be terminated by calling the Disconnect method. An application must close all connections it has created before it quits.

The Disconnect method takes no parameters and returns a boolean. If a connection is successfully terminated, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred. It fires an OnClose event to notify that the connection has been closed.

Calling this method is equivalent to setting the Action property to ACTION_DISCONNECT.

Example

```
Result = FTPClient.Disconnect ()  
If Result = False Then  
    MsgBox "Unable to disconnect from server", 64, "Sample Program"  
End If
```

4.15 EventMode

Summary

Set transfer mode to event based transfer.

Syntax

Boolean EventMode ()

Description

The EventMode method sets the file transfer mode to event based. This means that one or more OnSend events (in case of a put or append operation) or one or more OnReceive events (in case of a get operation) will be fired. An OnSend event asks the application to provide up to a maximum number of bytes of data, which are to be saved in the remote file. OnSend events are generated until the application indicates that there is no more data to send. An OnReceive event delivers a portion of the data from the remote file to the application. Event based file transfer allows an application to move data to and from storage other than the local disk, such as through DDE or the clipboard.

The EventMode method takes no parameters and returns a boolean. If the transfer mode can be successfully set to event mode file transfer, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the TransferMode property to TRANSFER_MODE_EVENT.

This method can be called at any time except during a file transfer.

Note

Event based mode gives you greater control over the file transfer. If the data you wish to transfer is not currently in a file, you should use an event based file transfer. Recovery following a failed transfer is only supported in an event based transfer.

Example

```
Result = FTPClient.EventMode ()
```

4.16 FileMode

Summary

Set transfer type to direct transfer.

Syntax

Boolean FileMode ()

Description

The FileMode method sets the file transfer mode to direct file transfer. This means that the FTP Client ActiveX control takes care of all input and output (I/O) operations required to get a file from the server or to put or append a file onto the server. All the application needs to do is to set the LocalFile and RemoteFile properties or pass the local file and remote file names to GetFile, PutFile, or AppendFile so that they uniquely identify one local and one remote file before starting the file transfer. Once the FILE_ACTION_GET, FILE_ACTION_PUT or FILE_ACTION_APPEND value is assigned to the FileAction property or the GetFile, the PutFile or the AppendFile method is called, then the ActiveX control completes the file transfer without generating any events or involving the application in any other way.

The FileMode method takes no parameters and returns a boolean. If the transfer mode can be successfully set to direct file transfer, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the TransferMode property to TRANSFER_MODE_FILE.

This method can be called at any time except during a file transfer.

Note

Use this mode to transfer files and directories without any file I/O. Data recovery on failed transfer is not supported in this mode. For data recovery you must use an event based transfer.

Example

```
Result = FTPClient.FileMode ()
```

4.17 FwConnect

Summary

Connect to FTP server via a SOCKS firewall server.

Syntax

Boolean FwConnect (*FwServer*, *FwPort*, *Host*, *User*, *Password*, *Account*)

<i>FwServer</i>	String
<i>FwPort</i>	Integer
<i>Host</i>	String
<i>User</i>	String
<i>Password</i>	String
<i>Account</i>	String

Description

The FwConnect method establishes a connection to the FTP server via a firewall.

If the FTP Client machine is located on a different subnet than the remote FTP Server machine and the only form of communication between these two machines is through a firewall gateway, then the built-in firewall support of the FTP Client ActiveX control can be used to establish an FTP session.

The FwConnect method takes a firewall server name (*FwServer*), a firewall port (*FwPort*), a host name (*Host*), a user name (*User*), a password (*Password*) and an account (*Account*) as its parameters and returns a boolean. *FwServer* must be the name or internet address (in dotted decimal notation) of the firewall server. *FwPort* must be the firewall service port. *Host* must be the name or internet address (in dotted decimal notation) of the FTP server. If the *Host* is the internet address then the *FwAddrType* property must be set to FW_ADDR_IP4 and if it is a machine name then the *FwAddrType* property must be set to FW_ADDR_DNS, by default the *FwAddrType* property has the value FW_ADDR_IP4. The *FwUsername* and *FwPassword* properties must contain a valid user name and password to complete the login process. Some servers also require that *Account* contain the name of an account for the user. If the server does not require the name of an account for the user, then *Account* can be set to an empty string.

The application can also set the *FwAuthMethods* property if it wants to specify any authentication that needs to be negotiated before an actual connection is established. Only the Username/Password authentication is currently supported. If Username/Password authentication is specified the *FwUsername* and *FwPassword* properties should contain a valid username and password respectively. An authentication is only possible if the SOCKS version is 5; version 4 SOCKS server does not support any authentication protocols.

The version of the SOCKS server can be specified in the *FwSocksVer* property, if the server version is unknown then the application can set this property to both FW_VERSION4 and FW_VERSION5, and the Distinct FTP ActiveX control will automatically detect the SOCKS version and connect appropriately.

If a connection is successfully established, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

If the connection can be established, then the OnConnect event will be fired. If the connection cannot be established, then the OnError event will be fired. These events will occur before the next statement (i.e. the statement following the call to the FwConnect method) is executed. The application should set a flag in the OnConnect and OnError events, so that it can determine if the session has been

established or not. In addition, the application may want to display an error message in the OnError event to inform the user that the connection has not been established. Please check the reference page of the OnError event for a complete listing of error codes.

Calling this method is equivalent to setting the Action property to ACTION_FW_CONNECT.

Note

If you need to connect through an HTTP proxy server, you must set the proxy properties and call the Connect method instead.

Example

```
Result = FTPClient.FwConnect ("sparky.distinct.com", 1080, "speedy.distinct.com", "santa", "north pole", "sc001d")
If Result = False Then
    MsgBox "Unable to connect to server via a firewall", 64, "Sample Program"
End If
```

4.18 GetFile

Summary

Retrieve file from server.

Syntax

Boolean GetFile (*RemoteFile*, *LocalFile*)

RemoteFile String
LocalFile String

Description

Files can be transferred to and from the FTP server either directly or through events depending on the setting of the TransferMode property or the call to the EventMode or FileMode method.

The GetFile method takes a remote file name (*RemoteFile*) and a local file name (*LocalFile*) as its parameters and returns a boolean. For an event based file transfer, pass an empty string as the local file parameter. If the file was successfully transferred, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

To do a direct file transfer you need to call the FileMode method first. In direct mode, a call to the GetFile method transfers the remote file to the local file. The remote file and the local file are passed as parameters to this method. This feature allows an application to transfer complete files of any size without doing any I/O itself. The local file parameter passed must contain a fully qualified path name since there is no current local directory.

To do an event based transfer, you must call EventMode first. If you have binary data to transmit, set the UseVariant property to TRUE. This will fire the ONReceiveB event to deliver the data instead of OnReceive. When a file is transferred through events, then one or more OnReceive events will be fired. An OnReceive event delivers a portion of the data from the remote file to the application. Event based file transfer allows an application to move data to and from storage other than the local disk, such as through DDE or the clipboard. For event based file transfer, the local file parameter is ignored and can be left blank.

Calling this method is equivalent to setting the FileAction property to FILE_ACTION_GET.

Note

Prior to this command, the Ascii or Binary methods should be called.

Example

```
Result = FTPClient.GetFile ("test.exe", "c:\test.exe")  
If Result = False Then  
    MsgBox "Cannot get file from server", 64, "Sample Program"  
End If
```

4.19 GetFileWithRestart

Summary

Retrieve file from server.

Syntax

Boolean GetFileWithRestart (*LocalFile*, *RemoteFile*, *Marker*)

<i>RemoteFile</i>	String
<i>LocalFile</i>	String
<i>Marker</i>	String

Description

This method is used to allow the user to recover from a failed file transfer and to complete the file transfer from the point of failure. For this to happen, this function must be used to initiate the file transfer as well as to recover from the failure.

The GetFileWithRestart method takes a remote file name (*RemoteFile*), a local file name (*LocalFile*) and *Marker* as its parameters and returns a boolean. A marker is a value returned by the server to indicate a particular point in the file transfer.

If the file was successfully transferred, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

In direct mode, a call to the GetFileWithRestart method transfers the remote file to the local file. The remote file and the local file are passed as parameters to this method. This feature allows an application to transfer complete files of any size without doing any I/O itself. The local file parameter passed must contain a fully qualified path name since there is no current local directory. . In direct mode *Marker* is ignored but user should set *RetryCount* property before calling this function. *RetryCount* property controls the number of times Restart command will be sent to the server in case of abnormal termination of file transfer. Default value of RetryCount is 0.

For an event based file transfer, pass an empty string as the *LocalFile* parameter. If *Marker* is not NULL, the file is retrieved from this *Marker* position, the user is required to position the local file corresponding to the marker position before calling this function. When a file is transferred through events, one or more OnReceive and OnReceiveMarker events will be fired. An OnReceive event delivers a portion of the data from the remote file to the application. An OnReceiveMarker event delivers a marker from the remote server to the application. User is required to save the current file position and the received server marker. In case user wants to restart the file transfer from the middle, user needs to position the local file to the saved file position and use GetFileWithRestart function with the corresponding server marker string.

Note:

When doing recovery in an event based transfer it is the responsibility of the application to know the corresponding byte position of the block of data immediately before the last marker was received for the local file.

Example

```
Result = FTPClient.GetFileWithRestart ("c:\test.exe", "test.exe", "")
If Result = False Then
    MsgBox "Cannot get file from server", 64, "Sample Program"
End If
```

4.20 GetMultipleFile

Summary

Get multiple files/directories from server.

Syntax

Boolean GetMultipleFile (*LocalDir*, *Remote*, *Recursive*)

<i>LocalDir</i>	String
<i>Remote</i>	String
<i>Recursive</i>	Boolean

Description

This function transfers multiple files from the host machine to the local system. The function can transfer specific files or entire directories. If the *Recursive* flag is set to True then the files will be transferred recursively.

The *LocalDir* parameter is a directory name and the *Remote* is specified as comma-separated file or directory names. For all directories, the transfer is always recursive. Wildcards can be specified for the *Remote* parameter.

If the files were successfully transferred, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

The application can abort the entire operation by calling Cancel at any time during the GetMultipleFile function call. Files are always transferred in direct mode. If mode is set to event transfer, it is ignored.

Note

The Ascii or binary method needs to be called prior to this method.

Example

```
Result = FTPClient.GetMultipleFile ("c:\test", "test.exe, test1", True)
If Result = False Then
    MsgBox "Cannot get files from server", 64, "Sample Program"
End If
```

4.21 ListDir

Summary

List contents of current directory on server.

Syntax

Boolean ListDir (*Wildcards*)
Wildcards String

Description

The ListDir method lists the contents of the current remote directory. The setting of the ListType property or the call to the LongList or the ShortList method determines if a short or a long listing will be sent. For every entry in the remote directory, the OnList event is fired. If an application does not want to receive any more entries, then it can call the AbortDir method or set the DirAction property to DIR_ACTION_ABORT in response to this event. Please check the OnList reference page for more information.

The ListDir method takes a *Wildcards* parameter and returns a boolean. The *Wildcards* parameter is used to limit the directory entries reported during a directory list operation to just those files and subdirectories matching the given wildcard expression. Only one wildcard expression, such as "*.TXT", can be specified. It is not possible to specify composite wildcards, such as "*.EXE *.COM" or "*.* - *.TMP". If the *Wildcards* parameter is set to an empty string, then all files and subdirectories will be listed. This is identical to setting this parameter to "*.*".

If the contents of the current directory can be successfully listed, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the DirAction property to DIR_ACTION_LIST.

Example

```
Result = FTPClient.ListDir ("")
If Result = False Then
    MsgBox "Cannot list current directory", 64, "Sample Program"
End If
```

4.22 Login

Summary

Allows the user to login after the connection has been reinitialized.

Syntax

Boolean Login (*User, Password, Account*)

<i>User</i>	String
<i>Password</i>	String
<i>Account</i>	String

Description

The Login method allows the user to login after the connection has been reinitialized. Generally Reinit and Login function are used in pair to change the user on the same connection.

Example

```
Result = FTPClient.Reinit ()
If Result = False Then
    MsgBox "Cannot reinitialize", 64, "Sample Program"
Else
    Result = FTPClient.Login ("Santa", "northpole", "gifts")
    If Result = False Then
        MsgBox "Cannot login", 64, "Sample Program"
    End If
End If
```

4.23 LongList

Summary

Set list type to long listing.

Syntax

Boolean LongList ()

Description

Directory listings are generated by calling the ListDir method or by setting the DirAction property to DIR_ACTION_LIST. Until all files in the current remote working directory have been listed (or until the AbortDir method is called or the DirAction property is set to DIR_ACTION_ABORT), OnList events are generated for each file or directory in the remote directory. OnList events generated for a long listing may include any number of additional pieces of information to the lines.

The LongList method takes no parameters and returns a boolean. If the list type can be successfully set to long listing, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

The format of long directory listing entries varies from one FTP server to another. Fields, such as name, size, date, and time, may be displayed differently and may not be in the same order. It is up to the application to interpret the incoming lines. In most cases, the directory entry lines can simply be displayed to the user without any further processing.

To get a long listing, call the LongList method or set the ListType property to LIST_TYPE_LONG before calling the ListDir method or before setting the DirAction property to DIR_ACTION_LIST.

Calling this method is equivalent to setting the ListType property to LIST_TYPE_LONG.

This method can be called at any time except during a directory list operation.

Example

```
Result = FTPClient.LongList ()  
Result = FTPClient.ListDir ("")
```

4.24 ParentDir

Summary

Change to parent directory on server.

Syntax

Boolean ParentDir ()

Description

The ParentDir method changes your current working directory on the FTP server to its parent directory. This is the same as calling the ChangeDir("..") method or setting the Target property to ".." and setting the DirAction property to DIR_ACTION_CHANGE.

The ParentDir method takes no parameters and returns a boolean. If the current working directory can be successfully changed to its parent directory, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the DirAction property to DIR_ACTION_PARENT.

Example

```
Result = FTPClient.ParentDir ()  
If Result = False Then  
    MsgBox "Cannot change to parent directory", 64, "Sample Program"  
End If
```

4.25 Parse

Summary

Parses a line of directory listing.

Syntax

Boolean Parse (*ListLine*, *IsDir*, *Name*, *Size*, *DateTime*, *Permissions*)

<i>ListLine</i>	String
<i>IsDir</i>	Integer
<i>Name</i>	String
<i>Size</i>	Long
<i>DateTime</i>	String
<i>Permissions</i>	String

Description

This method parses the directory list information (specified in *ListLine*) and returns the *isdir*, *name*, *size*, *date/time* and *permissions* in *IsDir*, *Name*, *Size*, *DateTime*, *Permissions* parameters. *ListLine* contains the line returned by the server during a *ListDir* command. Parameter *IsDir* returns 1 if it is a directory otherwise it returns 0.

Since the directory listing information varies with different host types, the *HostType* function must be called to indicate the remote host type before this function is called.

Note

If the *HostType* property is set to *Autodetect* which is its default value, when the *Parse* method is executed it will automatically issue the *SYST* command to try to find out the server type for the remote ftp server. If the particular server does not support the *syst* command, then the *HostType* will automatically default to *UNIX*. If this is incorrect, you need to set the *HostType* again before you can correctly parse the directory list information.

Example

```
Result = FtpClient.Parse(Target, IsDir, Name, Size, DateTime, Permissions)
If Result = False Then
    Exit Sub
Else
    sResult = "IsDir: " & Str(IsDir) & "Name: " & Name & " Size: " & Str(Size) & " DateTime: "
    & DateTime & " Permissions: " & Permissions
    MsgBox (sResult)
End If
```

4.26 PutFile

Summary

Upload file to server.

Syntax

Boolean PutFile (*LocalFile*, *RemoteFile*)

LocalFile String
RemoteFile String

Description

Files can be transferred to and from the FTP server either directly or through events, depending on the setting of the TransferMode property (or the call to the EventMode or FileMode method).

The PutFile method takes a local file name (*LocalFile*) and a remote file name (*RemoteFile*) as its parameters and returns a boolean. For an event based file transfer, pass an empty string as the local file parameter. If the file was successfully transferred, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

In direct mode, a call to the PutFile method transfers the local file to the remote file. The local file and the remote file are passed as parameters to this method. This feature allows an application to transfer complete files of any size without doing any I/O itself. The local file parameter passed must contain a fully qualified path name since there is no current local directory.

When a file is transferred through events, then one or more OnSend events will be fired. An OnSend event asks the application to provide up to a maximum number of bytes of data, which are to be saved in the remote file. OnSend events are generated until the application indicates that there is no more data to send. Event based file transfer allows an application to move data to and from storage other than the local disk, such as through DDE or the clipboard. For event based file transfer, the local file parameter is ignored and can be left blank.

Calling this method is equivalent to setting the FileAction property to FILE_ACTION_PUT.

Example

```
Result = FTPClient.PutFile ("c:\test.exe", "test.exe")
If Result = False Then
    MsgBox "Cannot put file to server", 64, "Sample Program"
End If
```

4.27 PutFileWithRestart

Summary

Upload file to server.

Syntax

Boolean PutFileWithRestart (*LocalFile*, *RemoteFile*, *Marker*)

<i>LocalFile</i>	String
<i>RemoteFile</i>	String
<i>Marker</i>	String

Description

The PutFileWithRestart method takes a local file name (*LocalFile*), a remote file name (*RemoteFile*) and a *Marker* as its parameters and returns a boolean. If parameter *LocalFile* is specified, parameter *Marker* is ignored and the file is put from the beginning otherwise if *Marker* is not NULL the file is put from this marker position using restart command.

If the file was successfully transferred, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Files can be transferred to and from the FTP server either directly or through events, depending on the setting of the TransferMode property (or the call to the EventMode or FileMode method).

In direct mode, a call to the PutFileWithRestart method transfers the local file to the remote file. The local file and the remote file are passed as parameters to this method. This feature allows an application to transfer complete files of any size without doing any I/O itself. The local file parameter passed must contain a fully qualified path name since there is no current local directory.

For an event based file transfer, pass an empty string as the *LocalFile* parameter. When a file is transferred through events, one or more OnSend and OnReceiveMarker events will be fired. An OnSend event asks the application to provide up to a maximum number of bytes of data, which are to be saved in the remote file. OnSend events are generated until the application indicates that there is no more data to send. An OnReceiveMarker event delivers a marker from the remote server to the application. Format of the marker string is h-l=m, where h=high word of local file position, l = low word of local file position, m = server marker. User is required to save this marker string. In case user wants to restart the file transfer from the middle, the user needs to position the local file to the saved file position and use the PutFileWithRestart function with the corresponding server marker string.

Event based file transfer allows an application to move data to and from storage other than the local disk, such as data input through the keyboard, through DDE or the clipboard. For event based file transfer, the local file parameter is ignored and can be left blank.

Example

```
Result = FTPClient.PutFileWithRestart ("c:\test.exe", "test.exe", "")
If Result = False Then
    MsgBox "Cannot put file to server", 64, "Sample Program"
End If
```

4.28 PutMultipleFile

Summary

Upload multiple files/directories to server.

Syntax

Boolean PutMultipleFile (*Local*, *RemoteDir*, *Recursive*)

<i>Local</i>	String
<i>RemoteDir</i>	String
<i>Recursive</i>	Boolean

Description

This function transfers multiple files from the local machine to the remote host. The function can transfer specific files or entire directories. If the recursive flag is set to TRUE then the files will be transferred recursively.

The *RemoteDir* parameter is a directory name and *Local* is specified as comma-separated file or directory names. For all directories, the transfer is always recursive. Wildcards can be specified for the *Local* parameter.

If the file was successfully transferred, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

The application can abort the entire operation by calling Cancel at any time during the PutMultipleFile function call. All the files are transferred in direct mode only. Any other setting for mode is ignored.

Example

```
Result = FTPClient.PutMultipleFile ("c:\test.exe, c:\test1", "test", True)
If Result = False Then
    MsgBox "Cannot put files to server", 64, "Sample Program"
End If
```

4.29 PutUniqueFile

Summary

Upload file to server with a unique file name.

Syntax

Boolean PutUniqueFile (*LocalFile*, *RemoteFile*)

LocalFile String
RemoteFile String

Description

Files can be transferred to and from the FTP server either directly or through events, depending on the setting of the TransferMode property (or the call to the EventMode or FileMode method).

The PutUniqueFile method takes a local file name (*LocalFile*) and a remote file name (*RemoteFile*) as its parameters and returns a boolean. For an event based file transfer, pass an empty string as the local file parameter.

If the file was successfully transferred, then the method returns True and *RemoteFile* property is set to the unique file name given by the server; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

In direct mode, a call to the PutUniqueFile method transfers the local file to the remote file. The local file and the remote file are passed as parameters to this method. This feature allows an application to transfer complete files of any size without doing any I/O itself. The local file parameter passed must contain a fully qualified path name since there is no current local directory.

When a file is transferred through events, then one or more OnSend events will be fired. An OnSend event asks the application to provide up to a maximum number of bytes of data, which are to be saved in the remote file. OnSend events are generated until the application indicates that there is no more data to send. Event based file transfer allows an application to move data to and from storage other than the local disk, such as through DDE or the clipboard. For event based file transfer, the local file parameter is ignored and can be left blank.

Example

```
Result = FTPClient.PutUniqueFile ("c:\test.exe", "test.exe")
If Result = False Then
    MsgBox "Cannot put file to server", 64, "Sample Program"
End If
Str = FTPClient.RemoteFile
```

4.30 ReceiveB

Summary

Retrieve binary data.

Syntax

Long ReceiveB (*Buffer*, *Bytes*)

<i>Buffer</i>	Variant
<i>Bytes</i>	Long

Description

The ReceiveB method retrieves the binary data and passes to the application.

The ReceiveB method takes a buffer (*Buffer*) and the number of bytes to read (*Bytes*) as its parameters and returns the actual number of bytes retrieved. The *Buffer* parameter should be an array of bytes data type. The *Bytes* parameter specifies how many bytes should be retrieved.

If there are less data to be read than the requested *Bytes*, only the available data are read. This is reflected in the return value. If the returned value is less than the requested *Bytes*, it means there are less data available than requested. The application should also check to make sure what is the actual number of bytes read.

Example

```
Length = FTPClient.ReceiveB (Buffer, Bytes)
```

4.31 Reinit

Summary

Terminates the current user but allows any file transfer in progress to complete.

Syntax

Boolean Reinit ()

Description

The function terminates the user associated with the connection handle. It flushes all I/O and account information but allows any transfers in progress to be completed. Generally Reinit and Login functions are used together to change the current user on the same connection.

If connection is successfully reinitialized, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Example

```
Result = FTPClient.Reinit ()
If Result = False Then
    MsgBox "Cannot reinitialize", 64, "Sample Program"
Else
    Result = FTPClient.Login ("Santa", "northpole", "gifts")
    If Result = False Then
        MsgBox "Cannot login", 64, "Sample Program"
    End If
End If
```

4.32 Remote

Summary

Remote to remote file transfer.

Syntax

Boolean Remote (*RemoteId*, *SrcFile*, *DestFile*)

<i>RemoteId</i>	Long
<i>SrcFile</i>	String
<i>DestFile</i>	String

Description

The Remote method transfers a file from one remote FTP server to another remote FTP server.

The Remote method takes the connection id (*RemoteId*) of another FTP client, a source file name (*SrcFile*) and a destination file name (*DestFile*) as parameters and returns a boolean. Remote to remote file transfer needs two FTP clients. For example FTPClient1 and FTPClient2. The *RemoteId* specifies the connection id of the second FTP client involved in the remote to remote transfer (e.g. FTPClient2.Id). To transfer a file between machine A and machine B: connect FTPClient1 to machine A and connect FTPClient2 to machine B. Then call the Remote method. This will transfer a file from machine A to machine B. There is no **local I/O** involved in the transfer.

If the source file can be successfully transferred to the other remote FTP server, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling the Remote method is equivalent to setting the FileAction property to FILE_ACTION_REMOTE.

Example

```
Result = FTPClient.Remote(FTPClient2.Id, "test", "abc")
If Result = False Then
    MsgBox "Cannot perform the specified remote to remote operation", 64, "Sample Program"
End If
```

4.33 RemoteAppend

Summary

Remote to remote file append.

Syntax

Boolean RemoteAppend (*RemoteId*, *SrcFile*, *DestFile*)

<i>RemoteId</i>	Long
<i>SrcFile</i>	String
<i>DestFile</i>	String

Description

The RemoteAppend method appends a file on the FTP server to another remote FTP server.

The RemoteAppend method takes a connection id (*RemoteId*) of another FTP client, a source file name (*SrcFile*) and a destination file name (*DestFile*) as parameters and returns a boolean. Remote to remote file transfer needs two FTP clients. For example FTPClient1 and FTPClient2. The *RemoteId* specifies the connection id of the second FTP client involved in the remote to remote transfer (e.g. FTPClient2.Id). To transfer a file between machine A and machine B: connect FTPClient1 to machine A and connect FTPClient2 to machine B. Then call RemoteAppend. This will transfer a file from machine A to machine B. There is no **local I/O** involved in the transfer.

If the source file can be successfully transferred to the other remote FTP server, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling the RemoteAppend method is equivalent to setting the FileAction property to FILE_ACTION_REMOTE_APPEND.

Example

```
Result = FTPClient.RemoteAppend (FTPClient2.Id, "test", "abc")
If Result = False Then
    MsgBox "Cannot perform the specified remote to remote operation", 64, "Sample Program"
End If
```

4.34 RemoveDir

Summary

Remove directory on server.

Syntax

Boolean RemoveDir (*Target*)
Target String

Description

The RemoveDir method removes a subdirectory on the FTP server with the specified target subdirectory name.

The RemoveDir method takes a *Target* parameter and returns a boolean. If the target directory can be successfully removed, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the DirAction property to DIR_ACTION_DELETE.

Example

```
Result = FTPClient.RemoveDir ("test")
If Result = False Then
    MsgBox "Cannot remove the specified subdirectory", 64, "Sample Program"
End If
```

4.35 RenameDir

Summary

Rename directory on server.

Syntax

Boolean RenameDir (*Target*, *NewName*)

Target String

NewName String

Description

The RenameDir method renames a subdirectory on the FTP server with the specified target subdirectory name to the specified new subdirectory name. This new subdirectory name cannot be the same as any existing subdirectory or file name.

The RenameDir method takes a *Target* and a *NewName* as its parameters and returns a boolean. If the target directory can be successfully renamed to the new directory name, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the DirAction property to DIR_ACTION_RENAME.

Example

```
Result = FTPClient.RenameDir ("test", "abc")
If Result = False Then
    MsgBox "Unable to rename directory", 64, "Sample Program"
End If
```

4.36 RenameFile

Summary

Renames the specified file in the current directory on the FTP server.

Syntax

Boolean RenameFile (*Target*, *NewName*)

<i>Target</i>	String
<i>NewName</i>	String

Description

The RenameFile method renames the file in the current directory of the FTP server with the specified target file name to the specified new file name. This new file name cannot be the same as any existing subdirectory or file name.

The RenameFile method takes a *Target* and a *NewName* as its parameters and returns a boolean. If the target file can be renamed successfully to the new file name, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the FileAction property to FILE_ACTION_RENAME.

Example

```
Result = FTPClient.RenameFile ("test.exe", "abc.exe")
If Result = False Then
    MsgBox "Unable to rename file", 64, "Sample Program"
End If
```

4.37 SendB

Summary

Send binary data.

Syntax

Boolean SendB (*Buffer*, *Bytes*)

<i>Buffer</i>	Variant
<i>Bytes</i>	Long

Description

The SendB method sends the binary data over an established connection. The application should not pass more data than the maximum specified by the Bytes argument of the OnSendB event to this method.

The SendB method takes a buffer (*Buffer*) and the number of bytes to be sent (*Bytes*) as its parameters and returns a boolean. The *Buffer* parameter should be an array of bytes data type. The *Bytes* parameter must contain the number of bytes of data to send. If the data is successfully sent, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

Example

```
Result = FTPClient.SendB (Buffer, Bytes)
If Result = False Then
    MsgBox "Cannot send binary data", 64, "Sample Program"
End If
```

4.38 ShortList

Summary

Set list type to short listing.

Syntax

Boolean ShortList ()

Description

Directory listings are generated by calling the ListDir method or by setting the DirAction property to DIR_ACTION_LIST. Until all files in the current remote working directory have been listed (or until the AbortDir method is called or the DirAction property is set to DIR_ACTION_ABORT), OnList events are generated for each file or directory in the remote directory. OnList events generated for a short listing will only include the name of a subdirectory or of a file.

To get a short listing, call the ShortList method or set the ListType property to LIST_TYPE_SHORT before calling the ListDir method or before setting the DirAction property to DIR_ACTION_LIST.

The ShortList method takes no parameters and returns a boolean. If the list type can be successfully set to short listing, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value. The method also sets the LastResult property. The value of the LastResult property can be checked to determine if any error occurred.

Calling this method is equivalent to setting the ListType property to LIST_TYPE_SHORT.

This method can be called at any time except during a directory list operation.

Example

```
Result = FTPClient.ShortList ()  
Result = FTPClient.ListDir ("*")
```

4.39 StreamMode

Summary

Set transmission mode to stream mode.

Syntax

Boolean StreamMode ()

Description

The function changes the transmission mode to stream mode. When a connection is opened, it defaults to STREAM mode.

This method can be called at any time except during a file transfer.

Example

```
Result = FTPClient.StreamMode ()
```

5 Registry Entries

5.1 FTP Client Registry Entries

This information is provided to document the registry entries used by the FTP Client ActiveX.

The following registry entry is stored under the key below:

\HKEY_LOCAL_MACHINE\SOFTWARE\Distinct\DLLS\FTP32

Value	Type	Description
WaitForSlowConnectionInMS	REG_DWORD	<p>This registry entry is used to add a delay in accepting a data connection. This delay may be needed when making a connection to an FTP Server using dialup networking on Windows NT 4.0. If an error of -8 is being fired from the DirList method, or during file transfer then add this registry entry.</p> <p>To add a delay create this registry entry, and set the key value to delay time in milliseconds. The value should be between 1 and 5 depending on the connection. This entry does not exist by default.</p> <p>Note: To make this entry run the Registry Editor, regedt32.exe under Windows NT and regedit.exe under Windows 95/98. Go to the following location in the registry: \HKEY_LOCAL_MACHINE\Software\Distinct\DLLS\FTP32. From the Edit menu select New, and then Dword Value. Type in "WaitForSlowConnectionInMS" for the Value Name, then double click on the Value Name and enter the delay time. The FTP Client will now delay before accepting the data connection.</p>
BufferSize	REG_DWORD	<p>This registry entry is used to specify the default size of the buffer used for data transfer.</p> <p>To specify the size create this registry entry, and set the key value to buffer size in bytes. The value should be between 1024 and 32768. The default buffer size is 32768. This entry does not exist by default.</p>
CtrlBuffer	REG_DWORD	<p>This registry entry is used to specify the default size of the buffer used for the control connection.</p> <p>To specify the size create this registry entry, and set the key value to buffer size in bytes. The value should be between 128 - 4096 depending on the connection. The default buffer size is 1024. This entry does not exist by default.</p>
Timeout	REG_DWORD	<p>Specifies the default timeout in seconds used for the FTP connection. This is the timeout value used for various network operations. The default value is 20 seconds.</p>

