Website: www.chestysoft.com Email: info@chestysoft.com

csFTPQuick - Version 3.0
COM Object for Transferring files between servers using FTP

This is a COM object that can be used to transfer files between servers using FTP. It contains two classes. The first contains basic functionality to upload or download a single file, a list of files, or an entire folder with optional subfolders. This is provided as a quick and easy way to automate file handling from a script, such as ASP. The second class encapsulates the main FTP commands and allows more complex FTP operations to be called programmatically from a script.

A free, fully functional trial version of csFTPQuick is available. If you are reading this instruction manual for the first time, it is likely that you have just downloaded the trial version. The trial version has a built in expiry date that causes it to stop working after that time. This is the only difference in functionality between the trial and full versions. This means that you can fully test if this component is suitable for your application before considering whether to license the full version.

Version 3.0 is supplied as two different DLL files, one is 32 bit and the other 64 bit. Refer to the next section for more details of registration and component instantiation.

Using these Instructions

These instructions are contain a section on the simple FTP class covering the general properties used for making an FTP connection, uploading files and downloading files. Then there is a section describing the general FTP commands. For quick links follow one of the links below, the Table of Contents on the next page, or the Alphabetical Index at the end.

csFTPQuick will often run in a server side script such as ASP and it will be used to transfer files to and from another server. When these instructions refer to the "remote server" it means the FTP server. When they refer to the "local" machine or the "client" they mean the computer that is running the component.

Click on one of the links below to go directly to the section of interest:

Chestysoft, July 2018
www.chestysoft.com

TABLE OF CONTENTS

1. Registering the Component and Getting Started

1.1. Registration and Server Permissions

Before the component can be used the DLL file must be registered on the server. This can be done using the command line tool REGSVR32.EXE. Take care to use the correct version of this tool as there is a 64 bit version in the Windows\System32 folder and a 32 bit version in the Windows\SysWOW64 folder. The syntax is:

regsvr32 *dllname*

where *dllname* is the path and name of the DLL to register.

Chestysoft has a free utility that performs this function through a Windows interface which can be easier although the result is identical. This tool can be downloaded from the Chestysoft web site: www.chestysoft.com/dllregsvr/default.asp

There are two DLL files supplied in the zip archive, one for 32 bit systems and one for 64 bit. The 32 bit file is called csFTPQuick.dll (csFTPQuickTrial.dll for the trial version). The 64 bit file is called csFTPQuick64.dll (csFTPQuick64Trial.dll for the trial version). The 64 bit file cannot be used on 32 bit systems.

We suggest creating a folder specifically for component DLLs rather than using the Windows System folder as this makes them easier to manage and avoids the naming confusion on 64 bit systems.

The application that uses the component must have permission to read and execute the DLL. In a web application like ASP this means giving the Internet Guest User account Read and Execute permission on the file. This account must also have the appropriate permissions for any file reading, writing or deleting that the component is to perform. Read permission is required to read/open a file from disk. Write permission is required to create a new file and Modify is required to edit or delete an existing file. These permissions can be set in Windows Explorer and applied to either a folder or individual files.

As this is an FTP component it is possible a firewall will need special configuration to allow it to run.

1.2. Object Creation

In any script or programme that uses the component an object instance must be created. There are two classes inside the component, *FTP* is used for simple FTP uploading and downloading. *FTPCommands* is used for passing specific FTP commands to the server. The syntax in ASP to create both classes is as follows.

For the full 32 bit version:

```
Set FTP = Server.CreateObject("csFTPQuick.FTP")

Set FTPCommands = Server.CreateObject("csFTPQuick.FTPCommands")
```

For the trial 32 bit version:

```
Set FTP = Server.CreateObject("csFTPQuickTrial.FTP")

Set FTPCommands = Server.CreateObject("csFTPQuickTrial.FTPCommands")
```

For the full 64 bit version:

```
Set FTP = Server.CreateObject("csFTPQuick64.FTP")
```

```
Set FTPCommands = Server.CreateObject("csFTPQuick64.FTPCommands")
```

For the trial 64 bit version:

```
Set FTP = Server.CreateObject("csFTPQuick64Trial.FTP")

Set FTPCommands = Server.CreateObject("csFTPQuick64Trial.FTPCommands")
```

The object names are "FTP" and "FTPCommands", but any variable names could be used. It is not necessary to create a class instance unless that class is used in the script. It is unlikely that both would be used in the same script.

1.3. The Trial Version

The trial version of the component is supplied as a separate DLL called csFTPQuickTrial.dll (or csFTPQuick64Trial.dll), with class names of "csFTPQuickTrial.FTP" and "csFTPQuickTrial.FTPCommands". This trial version is fully functional but it has an expiry date, after which time it will stop working. The object can still be created after the expiry date but it cannot connect to an FTP server.

The expiry date can be found by reading the *Version* property of the FTP class.

Version - String, read only. This returns the version information and for the trial, the expiry date.

Example:

```
Set FTP = Server.CreateObject("csFTPQuickTrial.FTP")
Response.Write FTP.Version
```

Visit the Chestysoft web site for details of how to buy the full version - www.chestysoft.com

1.4. Using csFTPQuick with Component Services

A COM component can be added to a COM+ Application in Component Services, One reason to do this is to be able to run a 32 bit DLL on a 64 bit system. Another is to specify a Windows account to use the component to allow that component to access network files that would be unavailable if the component was called by the default internet guest user.

An online description of configuring Component Services is available here:

http://www.chestysoft.com/component-services.asp

1.5. System Requirements

Version 3.0 does not support earlier Windows operating systems. It requires Windows 2003 or later for a server or Windows XP or later for a desktop. It will not register or run on Windows 2000. We can still provide version 2 for any users of an older operating system.

2. The FTP Class

The FTP class is used for uploading and downloading single files, lists of files at a single location, or complete folders with or without sub folders. The host name, username and password are set first as properties and connection and disconnection is taken care of automatically.

2.1. Properties for FTP connection

csFTPQuick connects to a remote FTP server and it must use an existing FTP account on that server. The *HostName*, *UserName* and *Password* properties must be set first before calling any of the upload or download methods.

HostName	-	String. This is the host name of the remote FTP server. It might be a domain name, an IP address or on a local network it might be the computer name.
UserName	-	String. The user name of the FTP account.
Password	-	String. The password of the FTP account.

Example:

```
Set FTP = Server.CreateObject("csFTPQuick.FTP")
FTP.Hostname = "ftp.xxxxx.com"
FTP.Password = "thepassword"
FTP.UserName = "theusername"
```

Once these properties are set the script is ready to move on to transferring files.

The port can be set but it defaults to 21, the default for FTP.

Port	-	Integer. The port number used by the FTP server. If the default port is used there is no need to set this property. (Default = 21).

It is possible to choose between Active Mode and Passive Mode by setting the *ActiveMode* property. It defaults to false, or Passive Mode.

ActiveMode	-	Boolean. Specifies whether the connection uses Active Mode or Passive Mode. (Default = false).

2.2. Uploading by FTP

There are three methods for uploading files, *Upload*, *UploadList* and *UploadFolder*. None of them take any parameters and all the appropriate settings are held in properties which must be set before calling the upload method.

Upload is for a single file. *UploadList* is for a number of files that are to be sent to a single folder on the server. The files are added to a list before calling the method. *UploadFolder* can upload the entire contents of a folder on the client and it can optionally include sub folders.

An exception will be raised if any of the upload methods fail to complete.

2.2.1. File Upload Methods

Upload	-	Uploads the file specified by *LocalFile* to the folder on the FTP site specified by *RemoteFolder*. It will be renamed to the name specified by *RemoteFile* if this property is not an empty string.

UploadList - Local files are specified by calling *AddLocalFile* and *UploadList* will upload all of them to *RemoteFolder*. They will be saved using their original name.

UploadFolder - The entire contents of *LocalFolder* will be uploaded to *RemoteFolder*. If *IncludeSubFolders* is true any sub folders will also be uploaded and they will automatically be created on the server if they do not already exist.

If a file by the same name already exists on the server it will always be overwritten.

AddLocalFile *FileName* - Adds a file to the list of files that will be uploaded using *UploadList*. *FileName* is a string and is the physical path to the file on the client.

ClearLocalFiles - Clears the list of local files that will be uploaded.

2.2.2. File Upload Properties

The following properties must be set before calling one of the upload methods. Some of these properties are reused for file downloading.

LocalFile - String. Physical path to the file which will be uploaded by *Upload*.

RemoteFile - String. File name to be used when saving the uploaded file with *Upload*. It is a name and extension with no path. If it is an empty string the file will use its existing name. (Default = empty string).

RemoteFolder - String. Folder on the FTP server, relative to the root, where files will be uploaded. In the case of *Upload* and *UploadList* the files are saved directly into this folder. If *UploadFolder* is used with *IncludeSubFolders* set to true sub folders will be created under this remote folder. The remote folder will be created if it does not exist.

LocalFolder - String. Folder on the local machine that will be uploaded by *UploadFolder*.

IncludeSubFolders - Boolean. When true, *UploadFolder* will also upload sub folders. (Default = false)

IncludeHiddenFiles - Boolean. Determines whether *UploadFolder* will include hidden files. (Default = false)

IncludeSystemFiles - Boolean. Determines whether *UploadFolder* will include system files. (Default = false)

Binary - Boolean. The FTP transfer type. When true files are uploaded as binary files and are unchanged. When false, files are uploaded as ASCII and formatting will be converted to local equivalents. Files will usually be uploaded as binary. (Default = true).

LocalFileCount - Integer, read only. Returns the number of files listed by *AddLocalFile*.

Section 3 contains some code fragments to demonstrate the syntax of the methods and properties.

2.3. Downloading by FTP

There are three methods for downloading files, *Download*, *DownloadList* and *DownloadFolder*. None of them take any parameters and all the appropriate settings are held in properties which must be set before calling the download method.

Download is for a single file. *DownloadList* is for a number of files that are to be downloaded from a single folder on the server to a single folder on the client. The files are added to a list before calling the method. *DownloadFolder* can download the entire contents of a folder on the server and it can optionally include sub folders.

An exception will be raised if any of the download methods fail to complete.

2.3.1. File Download Methods

Download - Downloads a single file, *RemoteFile* from *RemoteFolder* and saves it in *LocalFolder*.

DownloadList - Downloads the files listed by *AddRemoteFile* and saves them in *LocalFolder*. The folder containing the files on the FTP server is defined by *RemoteFolder* and only the name of the file is specified in the list.

DownloadFolder - Downloads all the files from the folder *RemoteFolder* and saves them in *LocalFolder*. If *IncludeSubFolders* is set to true all the sub folders and their contents will also be downloaded and the sub folders will be created locally if they do not already exist.

If a file by the same name already exists it will always be overwritten.

AddRemoteFile *FileName* - Adds a file to the list of files that will be uploaded using *DownloadList*. *FileName* is a string and the name of the file with no path information.

ClearRemoteFiles - Clears the list of remote files that will be downloaded.

2.3.2. File Download Properties

The following properties must be set before calling one of the download methods. Some of these properties are reused for file uploading.

RemoteFile - String. File to be downloaded by the *Download* method. It is a name only with no path information.

RemoteFolder - String. Folder on the FTP server, relative to the root, from where files will be downloaded. This property is always used to identify the location of a file for download.

LocalFolder - String. Folder on the local machine where downloaded files will be saved. *Download* and *DownloadList* will ony save to this folder but *DownloadFolder* will save a folder structure within the local folder if *IncludeSubFolders* is true.

IncludeSubFolders - Boolean. When true, *DownloadFolder* will also download sub folders. (Default = false)

RemoteFileCount - Integer, read only. Returns the number of files listed by *AddRemoteFile*.

Section 3 contains some code fragments to demonstrate the syntax of the methods and properties.

3. Code Samples for the FTP Class

Here are some code samples written for ASP in VBScript. Each script must create the object and set the connection properties, as shown in Section 2, so we will not repeat that here. Then the other properties must be set to specify files and folders. The FTP connection will be opened when the upload or download method is called, the file or files will be transferred and then the connection will be closed. There is no method to connect or disconnect and there is no functionality to keep the connection open and perform additional transfers.

3.1. Uploading a single file

Example 1 - Uploading a file to the root FTP folder:

```
FTP.LocalFile = "c:\images\test.jpg"
FTP.Upload
Response.Write "Finished"
```

LocalFile is set to the physical path of a local file. *RemoteFolder* is not set so the upload saves to the root on the FTP server. The Response.Write is simply to indicate that the script has completed. There is no feedback while the script is running and this may be a problem if a lot of files are transferred at once. Failure to connect or upload will raise an exception.

Example 2 - Upload a file to a sub folder:

```
FTP.LocalFile = "c:\images\test.jpg"
FTP.RemoteFolder = "sub1/sub2"
FTP.Upload
```

This will save the file in folder "sub2" which is under "sub1" which is under the FTP root. If this path does not exist it will be created.

Example 3 - Upload a file and rename it:

```
FTP.LocalFile = "c:\images\test.jpg"
FTP.RemoteFile = "newfile.jpg"
FTP.Upload
Response.Write "Finished"
```

This is the same as the first example except the file is renamed "newfile.jpg" when it is saved.

3.2. Uploading multiple files

Example 1 - Uploading multiple files with *UploadList*:

```
FTP.AddLocalFile "c:\files\file1.zip"
FTP.AddLocalFile "c:\files\file2.zip"
FTP.UploadList
Response.Write "Finished"
```

Local files are added with *AddLocalFile*. They can be in different folders but they will all be uploaded into the same folder on the FTP server. This destination will be the root in this code but it can be set to another folder by setting *RemoteFolder* as in Example 2 above.

Example 2 - Uploading an entire folder with *UploadFolder*:

```
FTP.LocalFolder = "c:\ftp\files"
FTP.UploadFolder
Response.Write "Finished"
```

LocalFolder is set to the folder that will be uploaded. In this case *IncludeSubFolders* has not been set so it takes its default value of false. *UploadFolder* transfers the files from the local folder but does not include any sub folders that are inside this folder.

Example 3 - Uploading a folder structure with *UploadFolder*:

```
FTP.IncludeSubFolders = true
FTP.LocalFolder = "c:\ftp\uploadedfiles"
FTP.UploadFolder
Response.Write "Finished"
```

The significant difference between this and the previous example is the use of the *IncludeSubFolders* property. Any sub folders under "c:\ftp\uploadedfiles" will also be uploaded and these sub folders will be created on the server if they do not already exist. Note that in this example the *RemoteFolder* property has not been set so the files in the folder defined by *LocalFolder* are uploaded to the root and any sub folders will be created under the root. There is no folder with the same name as *LocalFolder* created on the server. If the files are to be uploaded into a sub folder, this is done by setting *RemoteFolder*, for example:

```
FTP.RemoteFolder = "uploadedfiles"
```

This would cause the files to be uploaded to a folder called "uploadedfiles" which would be created if required.

3.3. Downloading a single file

Example 1 - Downloading a file to a local folder:

```
FTP.LocalFolder = "c:\downloads"
FTP.RemoteFile = "sample.zip"
FTP.RemoteFolder = "subfolder"
FTP.Download
```

The remote file to be downloaded is specified by its name and location, *RemoteFile* and *RemoteFolder*. *RemoteFile* is the name and extension of the file and *RemoteFolder* is the path relative to the root, which in this case is a single level sub folder called "subfolder". If *RemoteFolder* is not set it takes the default value of an empty string and the file will be downloaded from the FTP root.

The file is saved in the folder, *LocalFolder*.

3.4. Downloading multiple files

Example 1 - Downloading multiple files with *DownloadList*:

```
FTP.LocalFolder = "c:\downloads"
FTP.AddRemoteFile "file1.zip"
FTP.AddRemoteFile "file2.zip"
FTP.RemoteFolder = "sub1/sub2"
FTP.DownloadList
```

The files to be downloaded are listed by calling *AddRemoteFile* which takes the file name as a parameter. There is no path specified here and it is a method so there is no equals sign. If the files are in a sub folder, not the root, the relative path is specified in the *RemoteFolder* property. The files will

be saved locally in the folder defined by *LocalFolder*. *DownloadList* cannot be used to download files from multiple locations or save them to multiple locations.

Example 2 - Downloading the contents of a folder with *DownloadFolder*:

```
FTP.LocalFolder = "c:\downloads"
FTP.RemoteFolder = "subfolder"
FTP.DownloadFolder
```

The files in *RemoteFolder* will be downloaded to *LocalFolder*.

Example 3 - Downloading a folder structure with *DownloadFolder*:

```
FTP.LocalFolder = "c:\downloads"
FTP.RemoteFolder = "subfolder"
FTP.IncludeSubFolders = true
FTP.DownloadFolder
```

As Example 2 but *IncludeSubFolders* is set to true. This causes any sub folders below the remote folder to be downloaded and saved below the local folder. They will be created if required. There will be no folder with the name defined by *RemoteFolder* created on the local machine. It is the contents of this folder which are downloaded.

4. The FTPCommands Class

The FTPCommands class encapsulates the main commands used by FTP allowing the script to create and delete directories, rename and delete files, list the contents of a directory, change the current directory and transfer files in either direction. It is more versatile than the FTP class but some of the simpler file transfer operations will require more lines of code.

4.1. Connection and Disconnection

Connection is made to the FTP server with the *Connect* method and at the end of the session the *Disconnect* method should be called. *Disconnect* is automatically called when the object is freed from memory. The *Port* property can be set if a port other than the default 21 is to be used. The *ActiveMode* property can be set to enable either Active Mode (default) or Passive Mode.

Connect *HostName*, *UserName*, *Password* - Connects to the FTP server using the *HostName*, *UserName* and *Password* supplied as parameters.

Disconnect - Closes an existing FTP connection.

Port - Integer property. This defaults to 21 but if another value is assigned to this property before calling *Connect*, that port will be used instead.

ActiveMode - Boolean. Specifies whether the connection uses Active Mode or Passive Mode. (Default = true).

4.2. Administering Files and Directories

The following methods and properties control the creation and deletion of files and directories and also listing the contents of a directory.

GetCurrentDirectory - Returns the current directory as a string.

SetCurrentDirectory *DirectoryName* - This will set the current directory to *DirectoryName*.

CreateDirectory (*NewDirectory*) - This will create a new directory in the current directory, using the name *NewDirectory*. It is possible to create sub directories, separated by a "/" character in the name, but only under an existing directory. A Boolean return value indicates success or failure.

RemoveDirectory (*DirectoryName*) - This will delete the directory called *DirectoryName*, if it is empty. A Boolean return value indicates success or failure.

RenameFile (*OldName*, *NewName*) - This will rename the file called *OldName* to *NewName*, in the current directory. A Boolean return value indicates success or failure.

DeleteFile (*FileName*) - This will delete the file called *FileName* from the current directory. A Boolean return value indicates success or failure.

ListFiles - This method lists the files and directories in the current directory and stores their names as two lists which can be accessed by index. No parameters or return value.

FileListCount - Integer, read only. The number of files listed by *ListFiles*.

FileList (*Index*) - String, read only. The file specified by *Index* in the list produced by *ListFiles*. The first file has an *Index* of zero.

DirectoryListCount - Integer, read only. The number of directories listed by *ListFiles*.

DirectoryList (*Index*) - String, read only. The directory specified by *Index* in the list produced by *ListFiles*. The first directory has an *Index* of zero.

When files are listed using *ListFiles*, some attributes can be found including the size, date created, date last accessed and date last modified. These are also stored in indexed lists and they are the same size as *FileListCount*. Not all FTP servers return all these attributes.

FileSizeList (*Index*) - Integer, read only. The size of the file specified by *Index*, in bytes.

DateCreatedList (*Index*) - Date/Time, read only. The date created of the file specified by *Index*. If this value is not returned by the FTP server it will be zero.

DateAccessedList (*Index*) - Date/Time, read only. The date created of the file specified by *Index*. If this value is not returned by the FTP server it will be zero.

DateModifiedList (*Index*) - Date/Time, read only. The date created of the file specified by *Index*. If this value is not returned by the FTP server it will be zero.

4.3. Transferring Files

The FTPCommands class can upload and download single files, although as the connection remains open, each command can be repeated for multiple files. Files can be uploaded to and from disk and also as variant binary variables.

UploadFile *LocalName* - *LocalName* is the physical path to a local file and it will be uploaded to the current directory on the FTP server.

UploadBinary *BinaryFile*, *FileName* - *BinaryFile* must be a file stored in a variant array variable. This will be uploaded to the current directory on the FTP server and saved using the name *FileName*. *FileName* must be a file name with an extension but no path.

DownloadFile *LocalDirectory*, *FileName* - The file called *FileName* will be downloaded from the current directory on the FTP server and saved in the local directory called *LocalDirectory*. This is a directory path including a drive letter but no trailing backslash.

DownloadBinary *FileName* - Variant return value. The file called *FileName* in the current directory will be downloaded and returned as a variant array.

Files can be transferred either as binary or ASCII. The Boolean property Binary controls this behaviour.

Binary - Boolean property. When true, files are transferred in binary mode. When false they are transferred as ASCII files. (Default = true.)

4.4. Errors

Some of the methods of the FTPCommands class will raise exceptions if they fail. Others have a Boolean return value indicating success or failure. Generally it is the critical commands that raise exceptions, and these are *Connect*, *SetCurrentDirectory* and the upload and download methods. The methods *CreateDirectory*, *DeleteFile*, *RemoveDirectory* and *RenameFile* return a Boolean status.

5. Code Samples for the FTPCommands Class

Here are some code samples for the FTPCommands class written for ASP in VBScript.

5.1. Connecting and Disconnecting

Before any FTP activity can take place, a connection to the FTP server must be opened. This must be closed after use.

Example:

```
Set FTP = Server.CreateObject("csFTPQuick.FTPCommands")
FTP.Connect "ftp.xxxxxx.com", "theusername", "thepassword"
'
'Some FTP commands in here
'
FTP.Disconnect
```

The *Connect* method takes the host name, username and password as parameters. This will use port 21 by default as the Port property has not been set to a different value.

5.2. Listing the Files in a Sub Directory

The FTPCommands class has functionality to obtain a list of the files in a directory on the FTP server.

Example:

```
FTP.SetCurrentDirectory "users/aperson"
FTP.ListFiles
For I = 0 to FTP.FileListCount - 1
  Response.Write FTP.FileList(I) & "<br />"
Next
```

Use *SetCurrentDirectory* to set the directory on the FTP server. *ListFiles* will populate the list of files in the *FileList* array as well as the list of directories in the *DirectoryList* array. The number of files can be found by reading *FileListCount* and this can be used as a loop variable. Note that the index of *FileList* is zero based.

5.3. Downloading a File as a Variant Array

The variant data type is used in classic ASP to store raw data and this can be streamed to the browser using Response.BinaryWrite. It can also be stored in a binary database field or transferred between other ASP components, such as our image component csImageFile or our upload component csASPUpload.

Example:

```
Image = FTP.DownloadBinary("photo.jpg")
Response.ContentType = "image/jpeg"
Response.BinaryWrite Image
```

An image file on the FTP server can be downloaded and stored in an ASP variable and then streamed to the browser using Response.BinaryWrite.

6. Language Specific Issues

All the examples that are shown with the command descriptions use ASP and VBScript. The csFTPQuick component is a COM object and can be used in most COM enabled environments running on a Windows platform. We cannot begin to cover all the possible development environments here so instead we concentrate on ASP and in this section we show the syntax for Cold Fusion and Visual Basic.

6.1. Active Server Pages

ASP and VBScript is already covered in these instructions but the following points are worth noting.

Calls to methods do not use brackets, although from IIS 5 their use does not generate an error if there is only one parameter. For example:

```
FTP.AddLocalFile "C:\path\file.ext"
```

These instructions show methods without brackets surrounding the parameters, unless that method has a return value.

If the method has a return value that is assigned to another variable, the brackets are required. For example:

```
Response.Write FTP.DirectoryList(Index)
```

Assigning a property value requires an equals sign. Missing the equals sign also results in the error "Object doesn't support this property or method". The correct syntax is:

```
FTP.IncludeSubFolders = true
```

6.1.1. ASP with Javascript

We don't provide any examples of using ASP with other scripting languages, other than VBScript. We will mention the following about using Javascript with ASP.

Brackets are needed around function parameters.

The backslash character is used as an escape character in Javascript and two should be used together when a backslash is needed:

```
FTP.AddLocalFile("C:\\path\\file.ext");
```

6.2. Cold Fusion

In Cold Fusion, a COM object is created using the `<cfobject>` tag:

```
<cfobject action="create" name="ftp" class="csFTPQuick.FTP">
```

Each command must be placed inside a `<cfset>` tag and all method parameters must be enclosed by brackets:

```
<cfset FTP.Hostname = "ftp.xxxxx.com">
<cfset FTP.Username = "theusername">
<cfset FTP.Password = "thepassword">
<cfset FTP.LocalFile = "c:\path\file.ext">
<cfset FTP.Upload() >
```

Alternatively, the commands can be put inside a <cfscript> block:

```
<cfscript>
FTP.Hostname = "ftp.xxxxx.com";
FTP.Username = "theusername";
FTP.Password = "thepassword";
FTP.LocalFile = "c:\path\file.ext";
FTP.Upload();
</cfscript>
```

Cold Fusion does have its own built in FTP functionality in the form of the <cfftp> tag.

6.3. Visual Basic

For best results import the csFTPQuick type library into VB by selecting "Project" from the menu bar, then "References". The dialogue box will then show available type libraries. Scroll down to "csFTPQuick Library", check the box and click OK. This will add the "FTP" class from csFTPQuick to the Object Browser, making it available for early binding.

To create an instance of the object called "FTPObj" use the following code:

```
Dim FTPObj As FTP
Set FTPObj = CreateObject("csFTPQuick.FTP")
```

7. Revision History

The current version of csFTPQuick is 3.0.

New in Version 2.0:

The FTPCommands class has been added to allow more versatile control over the FTP process.

New in Version 3.0:

64 bit version released.

Changes to default behaviour to the FTP class. *ActiveMode* is now false by default to work with the default firewall settings in Windows 7. *UploadFolder* now does not include hidden or system files unless they are specifically included.

8. Other Products From Chestysoft

Visit the Chestysoft web site for details of other COM objects.

ActiveX Controls

csXImage	-	An OCX control to display, edit and scan images.
csXGraph	-	An OCX control to draw pie charts, bar charts and line graphs.

ASP Components

csImageFile	-	ASP component to resize and edit images.
csDrawGraph	-	Component to draw pie charts, bar charts and line graphs.
csASPUpload	-	Process file uploads through a browser.
csASPZipFile	-	Create zip files and control binary file downloads.
csFileDownload	-	Control file downloads with an ASP script.
csASPGif	-	Create and edit animated GIFs.

ASP.NET

csNetUpload	-	ASP.NET component for saving HTTP uploads.

Web Hosting

We can offer ASP enabled web hosting with our components installed. Click for more details.

9. Index of Commands – The FTP Class

10. Index of Commands – The FTPCommands Class