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

csFileDownload 3.0 - ASP File Download Component

This component allows file downloads to be controlled from within an ASP script. This is useful for restricting access to certain files and for record keeping. The component can load a file from disk on the server, from a network location or a remote URL and stream the file as binary data through the Response BinaryWrite command in ASP.

In addition there is a simple access code generation feature which can be used with the file download functions as part of a download verification system. There are also a number of file utility functions.

A free, fully functional trial version of csFileDownload is available. This trial version has a built in expiry date that causes the main functions 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 divided into a number of sections with the relevant methods and properties described in each. There are quick links to some sections below. A full Table of Contents is available on the next page and an index listing all commands in alphabetical order is included at the back for easy reference. The PDF version also has bookmarks for direct navigation to each heading.

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

Chestysoft, October 2013
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.

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 csFileDownload.dll (csFileDownloadTrial.dll for the trial version). The 64 bit file is called csFileDownload64.dll (csFileDownload64Trial.dll for the trial version). The 64 bit file cannot be used on 32 bit systems.

Chestysoft has a free utility that performs the registration function through a Windows interface instead of using regsvr32. This tool can be downloaded from the Chestysoft web site:
www.chestysoft.com/dllregsvr/default.asp

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 the 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 file handling. 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.

1.2. Object Creation

In any script or programme that uses the component an object instance must be created. The syntax in ASP is as follows.

For the full 32 bit version:

```
Set Download = Server.CreateObject("csFileDownload.Binfile")
```

For the trial 32 bit version:

```
Set Download = Server.CreateObject("csFileDownloadTrial.Binfile")
```

For the full 64 bit version:

```
Set Download = Server.CreateObject("csFileDownload64.Binfile")
```

For the trial 64 bit version:

```
Set Download = Server.CreateObject("csFileDownload64Trial.Binfile")
```

In each case the object name is "Download", but any variable name could be used.

1.3. The Trial Version

The trial version of the component is supplied as a separate DLL called csFileDownloadTrial.dll (or csFileDownload64Trial.dll). 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 be used to download files.

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

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

Example:

```
Set Download = Server.CreateObject("csFileDownloadTrial.Binfile")
Response.Write Download.Version
```

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

1.4. Using csFileDownload 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

On Windows 2008 and later it is necessary to "Allow intrinsic IIS properties" in the COM+ component properties. csFileDownload will run without this but the StreamFile method and some of the utility functions require IIS intrinsic properties.

1.5. System Requirements

csFileDownload 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. Controlling Downloads

The csFileDownload object controls file downloads by loading the file into memory from wherever it resides on the server, then streaming it to the browser. This allows code to be run before or after file transfer enabling form variables to be read, databases to be updated and any other record keeping that is required. The file to be downloaded does not need to be on a part of the server that is web shared, but the Internet Guest User must have read permission on that file.

The ASP script that controls the download does not return an HTML page. It must have the response buffer set to true, and the appropriate MIME type set using Response.ContentType. The example in the next section shows this.

The browser may display the file or it may ask the user if they want to open or save the file. The exact behaviour depends on the type of browser and its configuration.

2.1. The StreamFile method

The simplest way of sending a file to a browser involves calling the *StreamFile* method. The physical path of the file to be downloaded is specified by first setting the *FileName* property.

FileName	-	String. The physical path and name of the file to be streamed using the *StreamFile* method.
StreamFile	-	Sends the file specified by the *FileName* property to the browser as a binary stream.

Here is a sample script:

```
<%
  Response.Buffer = true
  Response.Expires = 0
  Response.ContentType = "application/x-zip-compressed"
  Set Download = Server.CreateObject("csFileDownload.Binfile")
  Download.FileName = "C:\download\sample.zip"
  Download.StreamFile
%>
```

This case shows a zip file being downloaded, so the ContentType is set accordingly. Setting the response buffer allows the download to be sent in smaller blocks instead of one big block and setting Response.Expires to zero stops the page being cached.

The *StreamFile* method builds the HTTP header and includes the file name and file size. Most browsers will prompt for this file name when saving the file. In the example shown that is "sample.zip". However, some browsers will use the name of the asp script instead.

An additional property, *PromptName*, can be set to send a different file name in the header. The previous example could be modified to prompt for the name "othername.zip":

```
Download.PromptName = "othername.zip"
Download.FileName = "C:\download\sample.zip"
Download.StreamFile
```

PromptName	-	String. The file name sent in the HTTP header when downloading using StreamFile, if different from the original file name.

As well as including the file name in the "Content-Disposition" header, the *StreamFile* method will specify whether the file should be displayed inline or as an attachment. Use the Boolean *Attachment*

property for this and set it to True to show the file as an attachment. The default value is False, causing the file to be displayed by the browser if possible. Not all browsers interpret this directive but Internet Explorer, Chrome and Firefox will.

Attachment - Boolean. Determines whether the *StreamFile* method will specify the download as an attachment or inline. (Default = false)

Behaviour does vary, not just between different browsers, but between browsers of the same type that have been configured differently by the user. It is important to check the behaviour of a download script in different browsers.

2.2. The FileData method

Another option is available to read file data which is more versatile. The *FileData* method takes the file name as an argument and returns an OLE variant containing the file data.

FileData(*FileName*) - Variant array return value. *FileName* is the path to the file.

Using *FileData* the previous example would become:

```
<%
  Response.Buffer = true
  Response.Expires = 0
  Response.ContentType = "application/x-zip-compressed"
  Set Download = Server.CreateObject("csFileDownload.Binfile")
  Response.AddHeader "Content-Disposition",
    "inline; filename=sample.zip"
  Response.AddHeader "Content-Length", Download.FileSize(
    "C:\download\sample.zip")
  Response.BinaryWrite Download.FileData("C:\download\sample.zip")
%>
```

These two examples have the same effect, but if the file data was to be stored in a database, for example, the *FileData* method would be used. The *StreamFile* method always adds the file name to the header, and always sends the data to the browser, and there may be occasions when this is undesireable.

The *FileData* method is more demanding on server memory and processing than *StreamFile* because it generates a variable that is as big as the file itself and so it should not be used with large files. IIS has a property called ASPBufferingLimit, or the Response Buffering Limit, which restricts the size of data that can be sent through BinaryWrite and it defaults to 4 MB, so *FileData* cannot be used with file sizes greater than this unless this property is changed. *StreamFile* sends the file in smaller chunks and is not restricted by ASPBufferingLimit.

2.3. Retrieving a File From a Remote Web Server

The csFileDownload component can get a file from a remote server using the URL and then save it or stream it to the browser. The following commands are used:

StreamFromURL *URL*, *FileName* - Streams the file at *URL* directly to the browser. *FileName* is the name sent to the browser.
SaveFromURL *URL*, *FileName* - Saves the file at *URL* to disk where *FileName* is the physical path of the destination.
URLData(*URL*) - Returns the file data as a variant array.

Example of saving a file from a remote URL:

```
Download.SaveFromURL "http://domain/directory/file.ext",
"C:\download\file.ext"
```

Example of streaming the file to the browser:

```
Response.ContentType = "application/x-zip-compressed"
Download.StreamFromURL "http://domain/directory/file.zip", "file.zip"
```

This example shows a zip file so the content type is set accordingly.

The above commands can specify a user name and password with the request. Passwords are sent as plain text if the server uses Basic Authentication. If the server uses Integrated Windows Authentication csFileDownload must be added to a COM+ application in Component Services, as described in Section 1.4 and a named account or the interactive user must be specified. Authentication will fail if the Service account is used.

Set the following properties before calling *StreamFromURL*, *SaveFromURL* or *URLData*.

URLUsername - String. Username to be passed with *StreamFromURL*, *SaveFromURL* or *URLData*.

URLPassword - String. Password to be passed with *StreamFromURL*, *SaveFromURL* or *URLData*.

The *HTTPUserAgent* property can be set to specify a user agent in the request header.

HTTPUserAgent - String. Value for the User Agent request header when *StreamFromURL*, *SaveFromURL* or *URLData* is called. This is null by default.

HTTPTimeout - Integer. Number of seconds before *StreamFromURL*, *SaveFromURL* or *URLData* will time out due to inactivity. A zero value is an indefinite time, and this is the default.

2.4. Verifying Completed Downloads

When files are downloaded using the csFileDownload component, it is possible to record whether the download was successful. This is done by using the Response.IsClientConnected command immediately after streaming the file to the browser. The earlier example is modified as follows:

```
<%
  Response.Buffer = true
  Response.Clear
  Response.ContentType = "application/x-zip-compressed"

  Set Download = Server.CreateObject("csFileDownload.Binfile")
  Download.FileName = "C:\download\sample.zip"
  Download.StreamFile
  Response.Flush
  If Response.IsClientConnected = true Then
    'The download was completed
    'Do something
  Else
    'The download was not completed
    'Do something else
  End If
%>
```

It is important to include Response.Flush immediately after the file was streamed. This effectively pauses the script while the data is in transit to the browser. If the connection is not maintained during

this time, Response.IsClientConnected will return false. It is also important to note that html output cannot be included at this stage and it is not possible to redirect the script. Only "hidden" server side processing can be done, but this can include database or text file manipulation to update records.

This method is not completely accurate. If the download is cancelled early enough neither of the options in the If statement will be reached. If the files is small (e.g. less than 200 KB) it will always appear to be a complete download. There is also no way to determine if the end user successfully saved the file after downloading.

2.5. Permissions and Accessing Remote Files

There are some important points to consider when working with files from a server side script. The script accesses files on the same server using the Internet Guest User account (IUSR_machine_name). This account frequently has limited default permissions so it may be necessary to adjust the permissions on files or directories that the script needs to work with.

Remote network files can be accessed by using the shared path or the UNC path but the csFileDownload component must first be registered as a COM+ application in Component Services, as described in Section 1.4.

It is possible that a firewall on the server computer can interfere with the access of remote files. This could affect network files or files read using the URL functions.

2.6. MIME Types

When files are streamed to a browser, it is important to specify the MIME type using Response.ContentType. The example above shows the MIME type for a zip file. There is a function available which obtains the MIME type, given the file extension. It extracts the information from the server system registry, which means the Internet Guest Account must have permission to access the registry. It should have this level of permission already because it needs it to find the registration details of the component.

GetMimeType(*Extension*) - Read only property, string return value. Extension is the file extension, with or without the period character. If there is no MIME type recorded for that file type the return value will be "application/unknown".

The code:

```
Response.Write Download.GetMimeType("zip")
```

would result in the output: "application/x-zip-compressed", assuming there is an entry for ".zip" in the registry and the script has permission to read the registry. Otherwise it would return "application/unknown".

It is usually better to find the correct MIME type first and hard code it into the script. It is unlikely that the GetMimeType method will work without making adjustments to the securtiy settings or using Component Services to give higher level access rights to the component.

3. The Access Code Function

The csFileDownload component has a built in function for generating access codes. This can be used in conjunction with file downloading, and is particularly useful if there is no database available on the server for maintaining a list of eligible users.

AccessCode (*String1*, *String2*, *IDNo*) - takes 3 strings as input and returns a 15 digit hexadecimal number. *String1* and *String2* can be anything. They could be name and email address, username and password, one could even be empty if required. The third value, *IDNo* is a number between 0 and 4294967295 (2^32). IDNo can be entered as an 8 digit hexadecimal number if it is enclosed in quotation marks and prefixed with a hash (#) or dollar ($) character, i.e. "#123456AB".

The returned value of *AccessCode* will always be the same for a given set of inputs, so a code can be given out on one web page and verified on another. The *IDNo* is hidden from the end user, so even if they have a copy of the component, they cannot predict the access code. *IDNo* should never be passed in a URL string or a form variable or displayed in any other way.

Here is an example:

A web page asks the user for their name and email address and passes these as form variables to a script which calculates an access code. The following lines will display the code:

```
<%
  Set Download = Server.CreateObject("csFileDownload.Binfile")
  Response.Write Download.AccessCode(Request.Form("Name"), _
    Request.Form("Email"), "#ABCDEF01")
%>
```

This creates an instance of the csFileDownload object called "Download". It then takes the two form variables and passes them to the *AccessCode* method along with a value for *IDNo*. If this was completed by a user named "Fred", with an email address of "fred@somewhere.com", the access code returned would be "66E78194DA6131F".

Another page asks the user for their name, email address and access code and passes these as form variables to a script which verifies the code. The following lines perform the verification:

```
<%
  Set Download = Server.CreateObject("csFileDownload.Binfile")
  If Download.AccessCode(Request.Form("Name"), _
    Request.Form("Email"), "#ABCDEF01") =  _
    Request.Form("AccessCode") Then
    'Code is correct
    'Do something
  Else
    'Code is not correct
    'Do something else
  End If
%>
```

This takes the name and email address that were supplied as form variables and passes them to the *AccessCode* method, along with the same value of *IDNo* used earlier. The return value is then compared with the code supplied by the user. Note that all the string values are case sensitive.
As with any password system, there will be ways of breaking it or bypassing it. This system has no guarantees, but there are many applications where this level of protection is appropriate.

4. File Utilities

There are a number of file utility functions included for convenience. They are not intended to be a comprehensive set, because standard ASP has the File System Object to cover most file utilities. These are the functions that are most likely to be useful while controlling file downloads.

CurrentDir - This property returns the actual path of the directory containing the script. It is complete with the trailing backslash character. It only works with ASP.

ParentDir(*Directory*) - *Directory* is a string value and must be a full directory path. The return value is the parent directory.

Example:

```
Response.Write Download.ParentDir(Download.CurrentDir)
```

This would display the parent directory to the one containing the current script.

DirName - String. This is the directory that will be listed in the *FileList* collection, described next. It is a full physical path and can include a filter in the file name.

Example:

```
Download.DirName = "C:\zipfiles\"
```

This will assign all the files in the "zipfiles" directory to the *FileList* collection.

```
Download.DirName = "C:\zipfiles\*.zip"
```

This will assign all the files with the extension .zip in the "zipfiles" directory to the *FileList* collection.

FileList - Collection of strings. When a directory is assigned to the *DirName* property this collection will be populated by a list of the files in that directory. As a Collection it can be accessed by index or in a For .. Each loop and it has a count property.

Example:

```
Download.DirName = "C:\zipfiles\*.zip"
For Each ZipFile in Download.FileList
  Response.Write ZipFile & "<br>"
Next
```

This would display all the zip files in the specified directory.

DirSortType - Integer enumeration. This determines the order of the files in the *FileList* collection. It must be set before the *DirName* property. Available values are 0 - alphabetical ascending, 1 - alphabetical descending, 2 - date order ascending, 3 - date order descending. The default is 0. For date sorting it is the last modified date that is used.

ScriptName - A read only property returning the current script name complete with extension. This only works with ASP.

FileSize(*FileName*) - *FileName* is the full path and filename of a file. The return value is the file size in bytes.

Delete(*FileName*) - This deletes the file *FileName*. Note that it is permanently deleted, NOT placed in the Recycle Bin.

Copy *OldName*, *NewName* - This copies the file *OldName* to the location and name

given by *NewName*. Full paths are required.

Rename *OldName*, *NewName* - This renames the file *OldName* to *NewName*. Full paths are required, and so renaming to a different directory is the equivalent of moving the file.

AppendToFile *FileName*, *NewLine* - This appends the string *NewLine* to the text file *FileName*. If the text file does not exist, it will be created if possible. The full physical path is required.

Example:

```
Download.AppendToFile Download.CurrentDir & "test.txt", "Hello"
```

This will append the line "Hello" at the end of a text file called test.txt which is in the same directory as the current script. If the file does not exist it will create it.

AppendToFile is the only command in this component for manipulating text files. It is useful for maintaining a simple log file containing download information. There is a full set of commands for dealing with text files in the built in File System Object.

All the file handling routines require that the Internet Guest Account has the appropriate permissions on the server, otherwise errors will result.

5. Revision History

The current version of csFileDownload is 3.0.

New in Version 2.0

Improvements to the StreamFile command to allow for use with larger files.
PromptName property added.
HTTPUserAgent property added.

New in Version 3.0

64 bit version released.

6. Other Products From Chestysoft

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

ActiveX Controls

csXImage	-	ActiveX control to display, edit and scan images.
csXGraph	-	ActiveX control to draw pie charts, bar charts and line graphs.
csXThumbUpload	-	Upload multiple files by HTTP or FTP with previews and image edits.
csXPostUpload		Uploads batches of files from a client to a server using an HTTP post.
csXMultiUpload	-	Select and upload multiple files and post to a server using HTTP.

ASP Components

csImageFile	-	Resize, create and edit images.
csDrawGraph	-	Draw pie charts, bar charts and line graphs in ASP.
csASPGif	-	Create and edit animated GIFs.
csIniFile	-	Read and Edit Windows style inifiles.
csASPUpload	-	Process file uploads through a browser.
csASPZipFile	-	Create zip files and control binary downloads.
csFTPQuick	-	ASP component to transfer files using FTP.

ASP.NET

csASPNetGraph	-	.NET component to draw pie charts, bar charts and line graphs.
csNetUpload	-	ASP.NET component for saving HTTP uploads.
csNetDownload	-	ASP.NET class to control file downloads.

Web Hosting

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

7. Alphabetical List of Commands