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

csFileDownload Version 2.0 - ASP File Download Component from Chestysoft

Introduction

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 an access code generation feature which can be used with the file download functions. There are also a number of file utility functions available.

The ActiveX file csFileDownload.dll should be registered on the server using REGSVR32.EXE. Regsvr32 is usually to be found in the Windows System folder, and it runs at the command prompt using the syntax:

`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. For more information on this, visit the Chestysoft web site, or follow: www.chestysoft.com/dllregsvr/default.asp

Object Creation

The code:

`Set Download = Server.CreateObject("csFileDownload.BinFile")` - creates an instance of the object called “Download”.

In the trial version, use:

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

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.

The StreamFile Method

The simplest and most efficient way of sending a file to a browser uses the *StreamFile* method. This involves setting the *FileName* property to the physical path of the file, and then using the *StreamFile* method to send it to the browser. 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
  Response.End
%>
```

This case shows a zip file being downloaded, so the ContentType is set accordingly. The *FileName* property is set using the physical path on the server. Note that the browser sees this as a binary file and there can be no HTML output in the same script.

The *StreamFile* method includes the file name in the header, so most browsers will prompt for the downloaded 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. A different name can be specified by setting the *PromptName* property. The above example could be modified to prompt for the name "newfile.zip":

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

The Attachment Property

As well as including the file name in the "Content-Disposition" header, the *StreamFile* method will also 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 display inline. Unfortunately not all browsers interpret this directive, Internet Explorer will but Netscape will not. See RFC 1806 for more details on Content-Disposition.

Example:

```
Download.Attachment = true
Download.StreamFile
```

This will cause some browsers, including Internet Explorer, to give the user a choice of saving or opening the file.

In summary, the *StreamFile* method will send a file to the browser. Related properties are *FileName*, *PromptName* and *Attachment*. *FileName* is required and must contain the physical path to the file. The behaviour of any browser or application reading a file streamed in this way can vary and cannot always be controlled from a server side script. The Response.Buffer property must always be set to true when using *StreamFile*.

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. The *StreamFile* example from above 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")
  Response.End
```

```
%>
```

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.

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. Unfortunately, it extracts the information from the server system registry, which would mean giving the Internet Guest Account permission to access the registry. This is not recommended for a live site but it could be done during development.

The syntax is: `GetMimeType(`*`Extension`*`)`

The code: `Response.Write Download.GetMimeType("zip")`

would result in the ouptput: `"application/x-zip-compressed"` if the script had permission to read the registry, and: `"application/unknown"` otherwise.

It is recommended to find the correct MIME type first and hard code it into the script.

Verifying Completed Downloads

A big problem with most server statistics is that when they count file downloads, they are counting the number of requests to the server, not complete downloads. By using the csFileDownload component, it is possible to count downloads more accurately. 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.Expires = 0
  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
  Response.End
%>
```

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. Only "hidden" server side processing can be done, but this can include database or text file manipulation to update records.

This method is still not completely accurate. If the download is cancelled early enough neither of the options in the If statement will be reached. It also has no way of determining if the end user successfully saved the file after downloading.

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. If FileName is an empty string the file name will be extracted from the URL instead. The response buffer must be set to true to use this command.

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 *FileName* parameter could have been an empty string in this case because the name is the same as in the URL.

The above commands can specify a user name and password with the request. It sends the passwords as plain text for Basic Authentication. It is not suitable for Integrated Windows Authentication. 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.

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. Failure to do this will lead to "cannot open file" errors. The permission used for accessing remote files is, by default, that of the user currently logged on to the server computer. This can be changed by editing the application properties once it is registered in Component Services.

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.

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.

The method:

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 pass 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.

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 component to cover most file utilities. These are the functions 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.

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. It is a full physical path and can include a filter in the file name:

```
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.

Scriptname - A read only property returning the current script name complete with extension.

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*. Again, 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 server 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.

Version - This is a read only property showing the version of the csFileDownload.dll file. In the case of the trial component, it shows the expiry date.

Revision History

The current version of csFileDownload is 2.0

New in Version 2.0:

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

Other Components From Chestysoft

Visit the Chestysoft web site for details of other components.

ASP Components

csASPUpload - Upload files through a browser form.
csASPZipFile - Create zip files and control binary downloads.
csImageFile - Resize and manipulate JPG, GIF, PNG, BMP, TIF and WBMP images.
csDrawGraph - Draw pie charts, bar charts and line graphs with an ASP script.
csASPGif - Create and edit animated GIFs.
csIniFile - Use Windows style inifiles in your ASP applications.

ActiveX Controls

csXImage - Control to display, edit and scan images.
csXGraph - Control to display pie charts, bar charts and line graphs.
csXPostUpload - Control to upload batches of files to a server using HTTP.
csXMultiUpload - Select and upload multiple files and post to a server using HTTP.

ASP.NET Components

csASPNetGraph - A .NET component to draw pie charts, bar charts and line graphs.
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.

Chestysoft, August 2006.
www.chestysoft.com