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

csASPUpload 3.1 - ASP File Upload Component

This ASP component is used to save files that have been attached to HTML forms using the <input type="file"...> tag. The files can be saved to disk on the server or extracted in a binary form for saving in a database or for passing to an image manipulation component.

Multiple files can be saved and functionality is provided to read the form variables, which cannot be read using the usual ASP request object following a file upload. Some file utility functions have been provided.

A separate class provides some additional HTTP functionality. This allows files to be transferred between servers, both by uploading and downloading. Form variables can be used and there is support for Windows Authentication.

A free, fully functional trial version of csASPUpload 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.

The component contains two classes, one for saving uploaded files (as well as the utility functions) and another for transferring files by HTTP. These classes have separate sections in the instructions and a separate alphabetical command list at the end.

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

Chestysoft, April 2016.
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 will be two DLL files supplied in the zip archive. One is for 32 bit systems and the other is for 64 bit. The 32 bit file is called csASPUpload32.dll (csASPUpload32Trial.dll for the trial version). The 64 bit file is called csASPUpload64.dll (csASPUpload64Trial.dll for the trial version). The 64 bit file cannot be used on 32 bit systems.

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

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 an image 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. There are two classes inside the component, Process is used for saving uploaded files and for the file utility functions. HTTP is used for the file transfer functionality. The syntax in ASP to create both classes is as follows.

For the full 32 bit version:

```
Set Upload = Server.CreateObject("csASPUpload32.Process")

Set Http = Server.CreateObject("csASPUpload32.HTTP")
```

For the trial 32 bit version:

```
Set Upload = Server.CreateObject("csASPUpload32Trial.Process")

Set Http = Server.CreateObject("csASPUpload32Trial.HTTP")
```

The object names are "Upload" and "Http", but any variable names could be used. It is not necessary to create a class instance unless that class is used in the script.

For the full 64 bit version:

```
Set Upload = Server.CreateObject("csASPUpload64.Process")

Set Http = Server.CreateObject("csASPUpload64.HTTP")
```

For the trial 64 bit version:

```
Set Upload = Server.CreateObject("csASPUpload64Trial.Process")

Set Http = Server.CreateObject("csASPUpload64Trial.HTTP")
```

In these instructions we shall show the 32 bit component in any example code.

1.3. The Trial Version

The trial version of the component is supplied as a separate DLL called csASPUpload32Trial.dll (or csASPUpload64Trial.dll). This trial version is fully functional but it has an expiry date, after which time it will stop working.

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

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

Example:

```
Set Upload = Server.CreateObject("csASPUpload32Trial.Process")
Response.Write Upload.Version
```

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

1.4. Limiting the Size of a File Upload

Windows 2003 (IIS 6) introduced a property that limits the amount of data that can be received using a POST operation, and this has a relatively low default value. This property can be changed by editing the metabase.xml file and it is called AspMaxRequestEntityAllowed. If a file is uploaded that is larger than specified by this property an error will result.

In IIS 7 and later, if Friendly Names are used, it will be called Maximum Entity Requesting Body Limit. It can be found by opening the ASP Properties and expanding Limits Properties.

1.5. Using csASPUpload with Component Services

The 32 bit version of csASPUpload can be used on 64 bit systems if it is added to a COM+ Application. An online description of configuring Component Services is available here:

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

On Windows 2008 and later it is important to "Allow intrinsic IIS properties" in the COM+ component properties.

Component Services provides some additional configuration options and also specifies the Windows account that will run the component.

2. Saving Uploaded Files and Reading Form Variables

All the methods and properties described in this section use the Process class, which is instantiated as described in section 1.2 above, using the class name "csASPUpload32.Process" (or the variations described previously). Saving file uploads and reading form variables can only be done when the Process class is called from an ASP script.

2.1. POSTing a File in an HTML Form

The HTML code used to attach a file for upload is shown below. It is important to set the enctype attribute as shown. Only a single file can be selected with the <input type="file"> tag and to upload multiple files, multiple tags must be used. Form variables can be included in the usual way.

```
<form method="post" action="filesave.asp" enctype="multipart/form-data">
<input type="file" name="filesent">
<input type="submit" value="Upload File">
</form>
```

In this example the script "filesave.asp" will receive the file and will be the script that uses the csASPUpload component.

2.2. The Read Method

The *Read* method must be called to parse the file upload. This must be done before any files can be saved or the form variables accessed. The properties controlling progress or saving directly without holding the file in memory must be set before calling *Read*.

Read	-	Call this method to process the file upload. It has no parameters and no return value.

Note that this method has been introduced in csASPUpload in version 3.0. Previous versions processed the upload when the component was created. Any script written the older version must be changed to include the *Read* method.

Calling *Read* in the same script as Request.Form will cause an error. Use the properties and methods described below to read form variables.

2.3. Form Variable Properties and Methods

The form variables cannot be read using the Request.Form collection, so properties are included specifically to access the form variables. Do not use any Request.Form commands on the same ASP page as the csASPUpload *Read* method as an error will be generated.

VarCount	-	Integer, read only. The number of variables received.
Value(*VariableName*)	-	String, read only. The value of a named variable, *VariableName*.

Example:

```
Response.Write Upload.Value("UserName")
```

This displays the value of the form variable "UserName".

VarByIndex(*index*)	-	String, read only. The variables are stored in a zero based array and can be retrieved using the index.

Example:

```
Response.Write Upload.VarByIndex(0)
```

This displays the value of the first variable.

VarName(*index*)	-	String, read only. Returns the name of the variable given its index.

Example:

```
Response.Write Upload.VarName(0)
```

This displays the name of the first variable.

Where variable names are duplicated the value returned by *Value* or *VarByIndex* will be a delimited string. By default this will be comma delimited but a different character or string can be specified in the *VarDelimiter* property.

VarDelimiter	-	String. The string used to delimit duplicated variables.

2.4. File Properties

File properties are also stored as zero based arrays. When a single file has been uploaded the index will always be zero but it must be included for correct syntax.

FileQty	-	Integer, read only. Returns the number of files received. This property is important as a check that files have been received, and for looping when multiple files are uploaded.

Example:

```
If Upload.FileQty = 0 Then
  Response.Write "No files received"
End If
```

FileName(*index*)	-	String, read only. The name of the file complete with extension.
Extension(*index*)	-	String, read only. The file extension without the period character.
NameOnly(*index*)	-	String, read only. The file name without the extension.
NewName(*index*)	-	String, read only. If the file is renamed during saving to prevent overwriting, this property is set to the new file name. See also the section on *OverwriteMode*.
ContentType(*index*)	-	String, read only. The content encoding type e.g. “image/gif”.
FileSize(*index*)	-	Integer, read only. The size of the file in bytes.

Example:

```
<p>The file size is <%= Upload.FileSize(0) %></p>
```

This would display the size of the first file in the array.

FileData(*index*) - Variant, read only. The file as binary data. This is used when saving to a database or passing the file straight to our csImageFile component. It could also be used with the Response.BinaryWrite method to stream the file back to the browser.

FileAsBase64(*index*) - String, read only. The file converted into a base64 encoded string. This can be used for saving into a database text field. Unlike FileData it returns text so it can be used with INSERT INTO in an SQL statement. Note that it does not include any content type information.

Example using FileData:

```
RSet("Image") = Upload.FileData(0)
```

This line would write the first file in the array to the database field “Image”, assuming a recordset called “RSet” had been opened. In MS Access, the data type for the field would be ”OLE Object”.

Example using FileAsBase64:

```
<img src="<%= Upload.FileAsBase64(0) %>" />
```

This would display an uploaded JPG file in an image tag. The content type must be included.

Note that when the form allows only one file to be uploaded, the index will always be zero. If the form is submitted with no file attached, i.e. the input field is blank, no file will be added to the *FileQty* total. Use *FileQty* to loop through arrays and to prevent index errors. Use *FileSize* to check that a file actually contains data.

2.5. The FileSave Method

The uploaded file is saved to disk on the server using the *FileSave* method.

FileSave *Filename*, *index* - *Filename* is the physical path and file name to be saved. *index* is the file to be saved where zero is the first or only file to have been uploaded.

Example:

```
Upload.FileSave "C:\temp\newfile.gif", 0
```

This saves the first file in the array using the path and name stated. Note that the index is after the comma and no brackets are required in VBScript.

Note: For any ASP script to save a file on the server, the Internet User Guest Account must have write permissions for that particular directory and modify permission to overwrite an existing file.

2.6. Saving Direct

The default way of handling file uploads is to hold each file in memory and then either save using *FileSave* or export as avariant using *FileData*. An alternative is to save the file directly to the server hard disk without loading it into memory first. This reduces memory usage and allows for much larger files to be handled. It is less flexible for handling multiple uploads and files cannot be exported using *FileData*.

Saving will be direct if the SaveDirect property is set to true before calling Read. The name and path of the saved file are set as properties, with an option to use the current file name. It is not possible to set each name individually of multiple files are uploaded.

The file properties, described in Section 2.4 can be used with files saved in this way.

SaveDirect - Boolean. When set to true, the Read method will save uploaded files to the server using the following properties to determine the path and file name. (Default = false)

SaveDirectPath - String. This is the physical path to the folder where the files are to be saved, including the trailing backslash. If this is empty and *SaveDirect* is true an error will be generated. (Default = empty string)

SaveDirectName - String. The file name to be used when saving direct. The full path and file name is made by combining *SaveDirectPath* and *SaveDirectName*. Only one file can be named in this way so it is not suitable for multiple uploads. (Default = empty string)

SaveDirectUseExistingName - Boolean. When true the file name used for saving direct is the existing name of the uploaded file. This can be used with *OverwriteMode*, described in the next section. (Default = false)

2.7. Overwrite Mode

OverwriteMode - Integer, 0, 1 or 2. This property determines what happens if an attempt is made to write to a file that already exists. The default value is 0.

OverwriteMode = 0	-	Any existing file will be overwritten.
OverwriteMode = 1	-	The new file will not be saved if an existing file has the same name.
OverwriteMode = 2	-	If the new file name matches an existing name the characters "~x" will be added at the end where "x" is the smallest number needed to make the name unique.

When an *OverwriteMode* of 2 is used, the file property *NewName* will be set to the name of the file that was actually saved.

Example:

```
Upload.OverwriteMode = 2
Upload.FileSave "C:\temp\newfile.gif" , 0
Response.Write Upload.NewName(0)
```

This will save the uploaded file as "newfile.gif", if that name is not already used. If a file with that name exists, it will save it as "newfile~1.gif" (or newfile~2.gif … etc.) The name actually used will be written out in the Response.Write statement.

If an alternative character to the '~' is needed this can be assigned to the property *OverwriteChr*.

OverwriteChr - String. The character or characters added before the number when a file is renamed using *OverwriteMode*. (Default = "~")

2.8. Simple Examples of Saving Files

Here are some simple examples of using both *FileSave* and *SaveDirect* to save files on the server. In each case the files are saved in the same directory as the script. Replace *Upload.CurrentDir* with the directory path to save somewhere else. Each example uses the *FileQty* property to check for at least one file before proceeding. This prevents array index errors. Server.MapPath could also be used to find the physical path on the server.

Example 1 - Single file

```
<%
Set Upload = Server.CreateObject("csASPUpload32.Process")
Upload.Read
If Upload.FileQty > 0 Then
  Upload.FileSave Upload.CurrentDir & Upload.FileName(0), 0
  Response.Write "<p>File saved</p>"
Else
  Response.Write "<p>No file received</p>"
End If
%>
```

Example 2 - Single file where uploads bigger than 1 MB are rejected.

```
<%
Set Upload = Server.CreateObject("csASPUpload32.Process")
Upload.Read
If Request.TotalBytes < 1048576 Then   '(no of bytes in 1 MB)
  If Upload.FileQty > 0 Then
    Upload.FileSave Upload.CurrentDir & Upload.FileName(0), 0
    Response.Write "<p>File saved</p>"
  Else
    Response.Write "<p>No file received</p>"
  End If
Else
  Response.Write "<p>File too big</p>"
End If
%>
```

Example 3 - Multiple uploads using *OverwriteMode* to rename duplicates.

```
<%
Set Upload = Server.CreateObject("csASPUpload32.Process")
Upload.Read
If Upload.FileQty > 0 Then
  Upload.OverwriteMode = 2
  For Index = 0 to Upload.FileQty - 1
    Upload.FileSave Upload.CurrentDir & Upload.FileName(Index), Index
  Next
  Response.Write "<p>Files saved</p>"
Else
  Response.Write "<p>No file received</p>"
End If
%>
```

Example 4 - Multiple uploads using *SaveDirect* and *OverwriteMode*.

```
<%
```

```
Set Upload = Server.CreateObject("csASPUpload32.Process")
Upload.SaveDirect = true
Upload.SaveDirectPath = "c:\files\"
Upload.SaveDirectUseExistingName = true
Upload.OverwriteMode = 2
Upload.Read
If Upload.FileQty > 0 Then
  Response.Write "<p>Files saved</p>"
Else
  Response.Write "<p>No file received</p>"
End If
%>
```

2.9. Matching Multiple Files With the HTML Tag

If multiple files are uploaded and the file must be matched with the tag on the HTML form, use the *TagNameIndex* property to find the index of the file.

TagNameIndex(*TagName*) - Integer, read only. This returns an integer which is the index needed to access the file and its details. *TagName* is the string value of the name attribute of the file tag and it is not case sensitive. If no file has been uploaded using that tag name the index will be -1.

This is where the tag name is defined in the form:

```
<input type="file" name="file1">
<input type="file" name="file2">
```

The code needed to save the second file, if present, might look like this:

```
If Upload.TagNameIndex("file2") > -1 Then
  Index = Upload.TagNameIndex("file2")
  Upload.FileSave Path & Upload.FileName(Index), Index
End If
```

Path is the path to the directory. This code checks that the second file has been attached and then finds the index. It uses this index to access the file name and save the file. It is dangerous to assume that the second file will always have an index of 1, because if no file is selected in the first tag, the file in the second tag will have an index of zero.

The reverse lookup is possible to find the tag name from the index of the uploaded file.

TagName(*Index*) - String, read only. This is the name attribute of the file tag given the index of the uploaded file. This is not always the index of the tag within the web page because a tag might not have a file selected.

2.10. Upload Progress

Web browsers do not usually display the progress of a file upload. csASPUpload includes the functionality to report the percentage of the upload that has been processed. This is done by setting and updating an application variable as the upload is processed. This application variable is unique to the user and it can be read and displayed in a pop up browser window.

ShowProgress - Boolean. This property is set to true to report upload progress. (Default = false)

ProgressID - String. This is the name of the application variable that will be used to record upload progress. It should be unique to the user. (Default = empty string).

Saving a file upload requires two pages. The form, which does not need any server side ASP, and the processing script, which is ASP and uses the csASPUpload component. When progress is being displayed, two additional pages are required. A pop-up will be generated when the upload form is submitted and this pop-up will display the progress in some way and it will contain an Ajax function which calls an ASP script which reads the application variable containing the progress percentage. The script that reads the application variable needs the value of ProgressID to be passed to it, because that is the name of the application variable. This ID value must also be passed to the script that saves the upload.

Sessions must be disabled in the website, otherwise the scripts cannot run concurrently and the progress is not updated until the upload is complete.

2.10.1. Example of Showing Upload Progress

The upload form should create a pop-up window when it is submitted:

```
<%
  Randomize
  ID = CInt(Rnd * 10000)
%>
<form method="post" action="save.asp?ID=<%= ID %>"
enctype="multipart/form-data" name="form1"
onsubmit="window.open('progressupdate.asp?ID=<%= ID %>', 'new',
'width=400,height=300')">
```

First, an ID is created that will be used by the various scripts. There are several ways to do this but a simple one is to create a random number. The *CreateGUID* method in csASPUpload could be used instead.

The uploaded file is selected and sent to a script called ‘save.asp’ and when it is submitted the page ‘progressupdate.asp’ is opened as a pop-up. ID is passed in the URL parameter to this pop-up page as well as the file saving script.

The upload processing script ‘save.asp‘ must set the appropriate properties:

```
Set Upload = Server.CreateObject("csASPUpload32.Process")
Upload.ShowProgress = true
Upload.ProgressID = Request.QueryString("ID")
Upload.Read
```

The pop-up, ‘progressupdate.asp’, contains the following:

```
<%
  ProgressID = Request.QueryString("ID")
%>
```

The ID is stored for later use. The next code is Javascript / Ajax to read and update the progress:

```
<script type="text/javascript">
function URLConnect(urlTo, mySend, x)
{
  var http = null;
  if(window.XMLHttpRequest)
```

```
    http = new XMLHttpRequest();
  else if (window.ActiveXObject)
    http = new ActiveXObject("Microsoft.XMLHTTP");
    http.onreadystatechange = function()
      {if(http.readyState==1)
         {
         }
         if(http.readyState == 4)
         {
           if(http.responseText != "")
           {
             x.innerHTML= http.responseText + "%";
           }
           else
           {
             x.innerHTML = "Upload complete"
           }
         }
      };
  http.open('POST', ''+urlTo, true);
  http.send(mySend);
  return false;
}

function updateProgress(){
x=document.getElementById("progress");
mySend="";
URLConnect('progress.asp?ID=<%= ProgressID %>', mySend, x);
}

setInterval("updateProgress()", 100)

</script>
```

This code reads the output from ‘progress.asp’ and uses it to update an HTML element. This value simply shows a percentage but there is scope to make something more graphical here. The update interval is 100 milliseconds and this could be changed to a longer period to reduce the number of requests to the server.

The HTML that is updated is:

```
<p><span id="progress">0</span></p>
```

Finally, the page ‘progress.asp’:

```
<%
ID = Request.QueryString("ID")
Response.Write Application(Request.QueryString("ID"))
%>
```

This script reads the application variable that has the same name as *ProgressID*. The csASPUpload component will clear this variable after the upload is complete.

2.10.2. Progress and Sessions

We have stated that session state must be disabled in order for the upload progress to be displayed, but this is not strictly true. The two scripts that report the progress, "progress.asp" and "progressupdate.asp" in our example above, can be removed from the session using the

EnableSessionState directive. The following code must be added to the first line of each of those scripts:

```
<%@ Language=VBScript EnableSessionState=False %>
```

This does not completely solve the problem and the scripts will still not run concurrently the first time a user uploads a file. The simple script that displays the application variable (progress.asp) must be called separately before the upload begins. Something similar to the Ajax code shown above could be used when the upload form first loads but a simpler solution is to use the HTTP object from csASPUpload. Add the following code to the start of the script containing the upload form:

```
Set HTTP = Server.CreateObject("csASPUpload32.HTTP")
Dummy = HTTP.DownloadData("http://full-url-here/progressvalue.asp")
```

This will copy the output of "progressvalue.asp" into a dummy variable, which is not used. The URL must be complete, not a relative path.

2.11. Code Examples

There are some downloadable examples available on the Chestysoft web site, including how to save files into a directory, saving to a database and how to resize an image using csImageFile. Follow this link for more details:

http://www.chestysoft.com/upload/default.asp

3. File Utilities

There are a number of file utility functions included in the Process class. They are not intended to be a comprehensive set, because ASP has the built in File System Object to cover most file utilities. These are the functions most likely to be useful while processing file uploads.

FileExists(*FileName*) - Returns a Boolean value. *FileName* is the physical path and file name of the file in question.

CurrentDir - Returns the physical 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 physical directory path. The return value is the parent directory.

Example:

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

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

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

Copy *OldName*, *NewName* - This copies the file *OldName* to the location and name given by *NewName*. The physical paths are required.

Rename *OldName*, *NewName* - This renames the file *OldName* to *NewName*. Physical paths are required. 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:

```
Upload.AppendToFile Upload.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 or to assist with testing. There is a full set of commands for dealing with text files in the built in File System Object.

CreateDir *Directory* - This creates a directory, where *Directory* is the physical path on the server, with or without the trailing backslash and including the drive letter. It has a boolean return value that is true if the directory was created or if it already exists.

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

CreateGUID - String return value. This returns a GUID for creating temporary files or providing a unique ID for use with the *ProgressID* property.

4. File Transfer Between Servers

All the methods and properties described in this section use the HTTP class, which is instantiated as described in section 1.2 above, using the class name "csASPUpload32.HTTP" (or "csASPUpload32Trial.HTTP" for the trial version of the component, or "csASPUpload64.HTTP" as appropriate). The HTTP class is not restricted to ASP and can be called from a wide range of COM enabled environments. It uses the Wininet.dll so Microsoft Internet Explorer must be installed on any machine where it is to be used.

Files can be uploaded (sent from the server running the component to a remote server) and downloaded (copied from a remote server to the server running the component) using HTTP. Uploaded files can be sent from disk or a binary variable. Downloaded files can be saved to disk or to a binary variable. Form variables can be sent with the upload and with the request for a file download. A username and password can be sent of the remote server requires authentication.

We have another component, csFTPQuick, that has FTP functionality for transferring files to and from a remote FTP server.

4.1. HTTP Properties

These properties apply to uploading or downloading a file. Some are set before an HTTP operation and two provide status information following an upload.

Username - String. The username to be sent with a request when the remote server requires authentication.

Password - String. The password to be sent with a request when the remote server requires authentication.

UserAgent - String. The User Agent header can be set to identify the caller of the request to the remote server. Note that some server configurations reject requests when the User Agent is empty. (Default = empty string)

TagName - String. Used with an upload, this is the equivalent of the name attribute of the file tag when a file is uploaded through the browser. (Default = empty string)

RequestMethod - String. Used with a download, this can be set to either "POST" or "GET" to specify the request method. It is not case sensitive and if an invalid value is given it will use the default. When the method is "POST" the form variables added with *AddFormVar* will be included. If the value is "GET" the form variables are not included and are not automatically added to the URL. (Default = "GET")

HTTPTimeOut - Integer. Timeout for the upload and download requests, in seconds. Zero is unlimited. (Default = 30)

ReturnFile - String, read only. This is the text of the web page returned by the remote server following an upload post. It can be used with Response.Write to display the complete page which is useful during debugging.

ReturnCode - Integer, read only. This is the HTTP code returned by the remote server following an upload post.

If Windows Authentication is used on the remote server, the csASPUpload component must be added to a COM+ application in Component Services and a named account must be used. The Network Service account is not sufficient for authentication.

4.2. HTTP Methods

There are two methods for posting uploads and two methods for requesting downloads. In each case, one method works with files on disk and other with files stored in memory as a variant binary variable.

UploadFile *URL*, *LocalFile* - This uploads the file at *LocalFile* to the address at *URL*. *URL* must be the full URL starting with "http://" or "https://" and it must be a server side script capable of saving a file upload. It has a Boolean return value which returns true when a 200 message is received from the server. An exception will be raised if the URL cannot be reached.

UploadData *URL*, *FileData*, *Name* - This uploads a file stored in the variant array variable *FileData* to the remote address at *URL*. Name is the file name and this is the equivalent of the file name field when uploading through a browser. The method has a Boolean return value which returns true when a 200 message is received from the server. An exception will be raised if the URL cannot be reached.

DownloadFile *URL*, *LocalFile* - This downloads a file from the remote *URL* and saves it to *LocalFile*. *URL* must be a full URL starting with "http://" or "https://". *LocalFile* is the physical path and file name on the same server as the script where the file will be saved. This method has no return value and raises an exception if it fails.

DownloadData *URL* - Variant return value. This downloads a file from the remote *URL* and returns it as a variant array. This can be used for further processing.

Example:

```
Response.ContentType = "image/gif"
Response.BinaryWrite HTTP.DownloadData("http://www.chestysoft.com/images/logo.gif")
```

This will download the logo from our web site and display it in the browser.

Form variables can be sent with uploads and with requests for downloads. Form variables are added by calling the *AddFormVar* method before the upload or download. In the case of downloads, the *RequestMethod* property must be set to "POST".

AddFormVar *Name*, *Value* - *Name* and *Value* are both strings and are the name and value of the form variable. Multiple variables are added by making multiple calls to *AddFormVar*.

ClearFormVars - This clears the list of form variables that have been added using *AddFormVar*.

Sometimes plain text is returned from a remote page and this can be difficult to extract in ASP when it is in a variant form. The *VariantToString* method can be used for a conversion. This may be used with the *DownloadData* method when a form has been posted and a text based result is returned.

VariantToString *InVar* - String return value. This will take a variant input parameter, such as the value returned by *DownloadData*, and return a string.

Example:

```
Return = HTTP.DownloadData("http://www.somesite.com/script.asp")
Response.Write HTTP.VariantToString(Return)
```

5. Upgrading from Version 2.0

csASPUpload version 3.0 represents some significant changes compared with previous versions. At version 2.0 and earlier the upload was processed when the Process class was instantiated. At version 3.0 the upload is not processed until the *Read* method is called. This allows some properties to be set before processing. Unfortunately if the component is replaced on an existing server, any scripts using the old component will need to be modified. Failure to update a script will lead to the *FileQty* property returning zero even when files have been uploaded and any attempt to call *FileSave* or *FileData* will give an index error.

In order to avoid breaking existing code when the component is replaced on a server the new dlls have new names and class IDs. The old version does not need to be removed and it will run alongside the new version without conflict. If existing code is ported to a new server using the new component, it will need to be modified to include the new class name and, when uploads are read, the call to the *Read* method must be added.

The new version of the component can handle larger files, if they are saved using *SaveDirect*. The maximum file size is around 2 GB. For files of a few megabytes or less, the default method of holding files in memory before saving is still acceptable but for larger files it is recommended to use the new method.

Version 3.0 has improved support for UTF-8 and Unicode characters in form variables and filenames.

Version 3.0 is supplied as two DLLs, one is 32 bit and the other 64 bit. The file name and class name includes the characters "32" or "64".

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 2.0 for any users of an older operating system but it cannot have any of the new features. The file sizes of the DLLs are also much larger than in version 2.0.

6. Revision History

The current version of csASPUpload is 3.0.

New in Version 1.1:

File utilities included for copying, renaming and deleting files and for obtaining the physical path on the server.
OverwriteMode introduced.

New in Version 1.2:

Maximum file size removed.
MaxKB property and Read method made obsolete.
User friendly error messages added for array index errors.

New in Version 1.3

TagNameIndex property added.

New in Version 2.0

HTTP class added.

New in Version 3.0

The Read method reintroduced.
SaveDirect and associated properties added.
ShowProgress and ProgressID added.
64 bit version released.

New in Version 3.1

FileAsBase64 added.

7. 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 downloads.
csFileDownload	-	Control file downloads with an ASP script.
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.

8. Alphabetical List of Commands - Process Class

These commands are used for saving uploaded files in ASP.

9. Alphabetical List of Commands - HTTP Class

These commands are used for transferring files by HTTP.