ASPFileSaver

File Upload Component

Website: www.aspphotoresizer.com

Email: info@aspphotoresizer.com

ASPFileSaver - File Upload Component

This is an ASP component for saving files that have been uploaded through a browser using the <input type="file"> tag. It supports uploads of single files and can either save this file or export it as a variant array for storing in a binary database field or exporting to our ASPPhotoResizer component. In ASP the Request.Form collection cannot be used when files are uploaded and so properties are provided for reading form variables.

A free, fully functional trial version of ASPFileSaver 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 control is suitable for your application before considering whether to license the full version.

Version 2.0 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 covering different types of functions available in ASPFileSaver. 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.

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

- Registering the Component and Getting Started
- Full Table of Contents
- Alphabetical Index of Commands
- Saving the File
- Form Variables
- Troubleshooting
- Upgrading from Version 1.0

ASPPhotoResizer.com, August 2014.
www.aspphotoresizer.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 in the zip archive. One is for 32 bit systems and the other is for 64 bit. The 32 bit file is called ASPFileSaver32.dll (ASPFileSaver32Trial.dll for the trial version). The 64 bit file is called ASPFileSaver64.dll (ASPFileSaver64.dll for the trial version). The 64 bit file cannot be used on 32 bit systems.

We can recommend a free utility that can perform registration through a Windows interface instead of using regsvr32. This tool can be downloaded here: 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 permission 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 Eplorer 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 Upload = Server.CreateObject("ASPFileSaver32.FileSave")
```

For the full 64 bit version:

```
Set Upload = Server.CreateObject("ASPFileSaver64.FileSave")
```

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

For the trial 32 bit version:

```
Set Upload = Server.CreateObject("ASPFileSaver32Trial.FileSave")
```

For the full 64 bit version:

```
Set Upload = Server.CreateObject("ASPFileSaver64.FileSave")
```

In these instructions we shall show the 64 bit component in any example code.

1.3. The Trial Version

The trial version of the component is supplied as a separate DLL called ASPFileSaver32Trial.dll (or ASPFileSaver64Trial.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.

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

Example:

```
Set Upload = Server.CreateObject("ASPFileSaver64Trial.FileSave")
Response.Write Upload.Version
```

Visit our web site to buy the full version - http://www.aspphotoresizer.com

1.4. The Upload Form

The HTML page that posts a file upload must contain a form that has the method and enctype attributes set as follows:

```
<form action="filesave.asp" method="post" enctype="multipart/form-data">
```

The action attribute must contain the address of the ASP script that will use the ASPFileSaver component to save the file. The method must be "post" and the enctype must be "multipart/form-data".

The file is selected using the input using the following form tag:

```
<input type="file" name="filesent">
```

This will display a text field and "Browse" button where the user can enter or select the local file on their system. Note that this field cannot be filled using Javascript. ASPFileSaver will only save a single uploaded file so if multiple Browse buttons are used, it will always be the first file that is saved. The name attribute is not used by ASPFileSaver but it must be present in the input tag.

1.5. 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 and it can be modified through IIS.

1.6. Using ASPFileSaver with Component Services

The 32 bit version of ASPFileSaver can be used on 64 bit systems if it is added to a COM+ Application. We recommend the following online description of configuring Component Services:

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 or Exporting the Uploaded File

The uploaded file can be saved to disk on the server using *SaveFile*. It can be exported as a variant array using *FileAsVariant*, and this would be used to send an image file to the ASPPhotoResizer component for processing or if the file was to be stored in a binary database field. Some properties are available to provide information about the file, such as *FileSize* and *Filename*.

The *Read* method must be called before any file can be saved or exported.

2.1. The Read Method

The Read method must be called to parse the file upload. This must be done before the file can be saved or the form variables accessed. The properties controlling progress or saving directly must be set before calling *Read*.

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

Note that this method has been introduced in ASPFileSaver in version 2.0. Previous versions processed the upload when the component was created. Any script written for the older version must be changed to include the *Read* method.

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

2.2. Saving to Disk

SaveFile *Filename* - This saves the uploaded file to the full physical path specified in *Filename*. The Internet Guest User must have Write permission on the destination directory to save the file and Modify permission to overwrite an existing file.

Example using a hard coded file name:

```
Upload.SaveFile "C:\files\newfile.jpg"
```

Use Server.MapPath to convert a virtual path to a physical path.

2.3. Exporting As Binary Data

FileAsVariant - Variant array, read only property. The uploaded file exported as a binary variable.

Example of saving the file to a binary database field called "Image":

```
RSet("Image") = Upload.FileAsVariant
```

Example of exporting the file to the ASPPhotoResizer component for further processing:

```
JPG.VariantIn = Upload.FileAsVariant
```

This assumes the instance of ASPPhotoResizer is called "JPG".

2.4. File Properties

The following properties provide information about the uploaded file.

ContentType - String. The MIME type specified during the upload.

Filename - String. The name of the file that was uploaded, complete with extension.

FileExtension - String. The file extension, without the period character.

FilenameNoExtension - String. The file name, with the extension removed.

FileSize - Integer. The size of the uploaded file in bytes. This property can be used to check if a file was uploaded, because it will always be zero when no file is uploaded.

NewName - 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*.

2.5. OverwriteMode

The *OverwriteMode* property provides a quick way to prevent files with duplicate names from being overwritten.

OverwriteMode - Integer, 0, 1 or 2. This property determines what happens if the *SaveFile* method attempts 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.SaveFile "C:\temp\newfile.gif"
Response.Write Upload.NewName
```

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*. This might be necessary on Windows 2003 or if URLScan is used because URLs containing this character may be blocked from downloading.

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

2.6. Saving Direct

The default approach to save file uploads is to hold the file in memory and then either save using *SaveFile* or export as a variant using *FileAsVariant*. An alternative provided in Version 2.0 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. Files saved in this way cannot be exported using *FileAsVariant*.

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.

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 the 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*. (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 earlier. (Default = false)

2.7. Some Examples

Example 1 - Saving a file with no error checking.

```
<%
  Set Upload = Server.CreateObject("ASPFileSaver64.FileSave")
  Upload.Read
  Upload.SaveFile Server.MapPath(".") & "\" & Upload.FileName
%>
```

This will save the file into the same directory as the script and it will use the existing file name for the saved file. If a file already exists by that name it will be overwritten. An error will be generated if the user submits the form without attaching a file.

Example 2 - Checking for an upload before saving.

```
<%
  Set Upload = Server.CreateObject("ASPFileSaver64.FileSave")
  Upload.Read
  If Upload.FileSize > 0 Then
    Upload.SaveFile Server.MapPath(".") & "\" & Upload.FileName
    Response.Write "<p>File saved.</p>"
  Else
    Response.Write "<p>No file received.</p>"
  End If
%>
```

This checks the *FileSize* property, which will be zero if no file was uploaded. The If..Then..Else statement will save the file if there is a file.

Example 3 - Using a form variable for the saved file name.

In this example a form variable is added to the form for the user to specify the name that will be used when the file is saved.

```
<input type="text" name="SaveName">
```

The script is similar to example 2 above except the variable name is used for the saved name and the file is saved into a sub directory. The previous examples have saved the files into the same directory as the script but a separate location would often be preferable.

```
<%
  Set Upload = Server.CreateObject("ASPFileSaver64.FileSave")
  Upload.Read
  If Upload.FileSize > 0 Then
    Upload.SaveFile Server.MapPath(".") & "\files\" &_
Upload.FormVariable("SaveName") & "." & Upload.FileExtension
    Response.Write "<p>File saved.</p>"
  Else
    Response.Write "<p>No file received.</p>"
  End If
%>
```

In practice, the file name would rarely be chosen like this and some error checking would be needed to make sure the name was valid. This example shows how to read a form variable and construct a name based on a variable and the existing extension.

The functions for reading form variables are described in the next section. Request.Form cannot be used with file uploads. Request.Querystring can be used as normal.

Example 4 - Using SaveDirect and OverwriteMode.

```
<%
  Set Upload = Server.CreateObject("ASPFileSaver64.FileSave")
  Upload.SaveDirect = true
  Upload.SaveDirectPath = "c:\files\"
  Upload.SaveDirectUseExistingName = true
  Upload.OverwriteMode = 2
  Upload.Read
  If Upload.FileSize > 0 Then
    Response.Write "<p>File saved.</p>"
  Else
    Response.Write "<p>No file received.</p>"
  End If
%>
```

2.8. Upload Progress

Web browsers do not usually display the progress of a file upload. ASPFileSaver 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 and so it could be set by using *CreateGUID* to generate a unique string. (Default = empty string)

CreateGUID - String return value. This returns a GUID which can be used for naming temporary files or for providing a unique ID for use with the *ProgressID* property.

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 ASPFileSaver 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.

2.8.1. 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. ASPFileSaver has a *CreateGUID* method which could also be used to create a unique string.

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("ASPFileSaver64.Process")
Upload.ShowProgress = true
Upload.ProgressID = Request.QueryString("ID")
Upload.Read
```

If the session ID was not used, the ID would be passed in the URL.

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 ASPFileSaver component will clear this variable after the upload is complete.

2.8.2. 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 form first loads but a simpler solution is to use the Microsoft XML HTTP component to send a request to the script. Add the following code to the start of the script containing the upload form:

```
Set XML = Server.CreateObject("Microsoft.XMLHTTP")
XML.Open "GET", "http://full-url-here/progressvalue.asp"
XML.Send
```

This will send a request to "progressvalue.asp" and then the next time it is called it will not be part of the session.

3. Form Variables

Form variables cannot be read using Request.Form, so the following properties must be used instead. *FormVariable* reads the variable given its name. *FormVariableCount* returns the number of variables in the form, and *FormVariableByIndex* returns the value given the index of the variable within the form.

The *Read* method must be called before using any of these properties.

FormVariable (*VariableName*) - String. The value of the variable where *VariableName* is the string name of the variable.

Example:

```
Response.Write Upload.FormVariable("User")
```

This will display the value for a form variable called "User".

FormVariableCount - Integer. The number of form variables in the form. This includes the name of the uploaded file.

FormVariableByIndex (*Index*) - String. The value of the variable specified by *Index*. *Index* represents the number of the variable within the form, where the first has an index of zero and the last has an index of *FormVariableCount* - 1.

Example of looping through all the form variables:

```
For I = 0 to Upload.FormVariableCount - 1
  Response.Write Upload.FormVariableByIndex(I) & "<br>"
Next
```

FormVariableName (*Index*) - String. Returns the name of the variable given its index.

Example:

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

This will display name of the first form variable.

Where variable names are duplicated the value returned by FormVariable or FormVariableByIndex will be a delimited string. By default this will be comma delimited but a different character or string can be specified in the FormVariableDelimiter property.

FormVariableDelimiter - String. The string used to delimit duplicate variables. (Default = ",")

4. Troubleshooting

This section covers some of the common causes for errors when using the ASPFileSaver component.

4.1. Failure to Register the Component

The component must be registered on the server as described in Section 1.1. If the component is not registered it will generate the error:

Server object, ASP 0177 (0x800401F3)
Invalid ProgID.

4.2. Incorrect Class Name

If the class name inside the Server.CreateObject command is incorrect it will generate exactly the same error as when the component is not registered. Take care to spell the class name correctly. Also, note that the class name is different for the trial version, compared with the full version and between the 32 bit and 64 bit versions. See Section 1.2 for more on the class names and object creation.

4.3. Insufficient Permission on the DLL

An ASP script runs under the Internet Guest User account, unless specifically configured otherwise. This account must have Read and Execute permission on the ASPFileSaver DLL file. Failure to do this will result in the error:

Server object, ASP 0178 (0x80070005)
The call to Server.CreateObject failed while checking permissions. Access is denied to this object.

4.4. Insufficient Permission to Save Files

The Internet Guest User account must have permission to write into the directory where the files are to be saved. Insufficient permissions will result in the following error:

ASPFileSaver64.FileSave (0x8000FFFF)
Error saving file. Check the user permissions.

Modify permission is required to overwrite existing files.

4.5. Using Request.Form in the Script

Request.Form commands must not be used in the same script as the ASPFileSaver component. The exact error generated will depend on where the Request.Form command appears in the script.

Section 3 describes how to read form variables.

4.6. Upload Too Big

IIS contains a property that limits the amount of data that can be received using a POST operation, and this defaults to the relatively low value of 200 KB. This property is called *ASPMaxRequestEntityAllowed* and it can be changed to a higher value in IIS, or by editing the metabase.xml file in Windows 2003.

5. Upgrading From Version 1.0

ASPFileSaver version 2.0 represents some significant changes compared with previous versions. At version 1.0 and earlier the upload was processed when the component was instantiated. At version 2.0 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 *FileSize* property returning zero even when files have been uploaded and any attempt to call *SaveFile* or *FileAsVariant* will give an 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, such as image files, 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. If the uploaded files are images which are to be passed into the ASPPhotoResizer component, the default method must be used.

Version 2.0 has improved support for UTF-8 and Unicode characters in form variables and filenames.

Version 2.0 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 2.0 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 1.0 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 1.0.

6. Alphabetical List of Commands