



Website: www.chestysoft.com

Email: info@chestysoft.com

csXThumbUpload - ActiveX Control to Preview and Upload Files Using HTTP or FTP

This ActiveX control lists the files in a folder on the client computer, showing the files either as icons or image previews. The user can then select one or more files and upload them to a remote server. Uploads can either be by HTTP or FTP.

Image files can be resized before the upload and some other edits are also available, such as a change of file format, colour depth or removal of metadata.

csXThumbUpload can be embedded in a web page (Internet Explorer only) using Javascript or VBScript, or it can be used within a COM enabled programming environment.

The trial version of the control shows a pop up message in between each upload. Otherwise it is fully functional allowing a complete evaluation of the control before buying the full version.

When uploads are made by HTTP there must be a server side script which is capable of saving the files. We supply free components for saving uploaded files in ASP and ASP.NET although these components are designed for use with csXThumbUpload only and will not save general file uploads. Other upload tools may be used if preferred.

Using These Instructions

These instructions are divided into a number of sections with quick links to some of the 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:

- [Installation and Getting Started](#)
- [Properties Controlling Appearance and Layout](#)
- [Properties Controlling Uploads](#)
- [Properties Controlling Image Editing](#)
- [Saving Uploads in the Server](#)
- [Alphabetical List of Commands](#)

TABLE OF CONTENTS

1	INSTALLATION AND GETTING STARTED	3
1.1	USING CSXTHUMBUPLOAD IN INTERNET EXPLORER	3
1.1.1	<i>Licensing and LPK Files</i>	3
1.1.2	<i>Parameters</i>	4
1.1.3	<i>Version</i>	4
1.2	USING CSXTHUMBUPLOAD IN VISUAL BASIC	4
1.3	OTHER DEVELOPMENT ENVIRONMENTS	4
1.4	THE TRIAL VERSION	5
1.5	SUPPORTED IMAGE FORMATS	5
2	PROPERTIES CONTROLLING APPEARANCE AND LAYOUT	6
3	PROPERTIES CONTROLLING UPLOADS	8
4	PROPERTIES CONTROLLING IMAGE EDITING	10
5	PROPERTIES CONTROLLING DISPLAYED TEXT	12
6	EVENTS	13
7	CODE EXAMPLES	15
8	SAVING FILES UPLOADED BY HTTP	17
8.1	ASP	17
8.2	ASP.NET	17
8.3	COLD FUSION	18
8.4	PHP	18
9	OTHER PRODUCTS FROM CHESTYSOFT	19
10	ALPHABETICAL INDEX OF COMMANDS	20

1 Installation and Getting Started

Running the installation executable will copy the files into either the default folder or a folder of your choice. It will also register the OCX and place a link to the instructions in the Start Menu. A sample HTML page is supplied and this can be reached from the Start Menu but it will not work until the *RemoteURL* property has been completed.

1.1 Using csXThumbUpload in Internet Explorer

The following code is used to create an instance of csXThumbUpload in a web page.

Full Version

```
<object id="upload" classid="CLSID:078B03B7-23A0-4EA6-A0A9-6E27EA68BE79" width="764" height="580"></object>
```

Trial Version

```
<object id="upload" classid="CLSID:A86861EE-577E-4C0D-BABB-E09920703ED0" width="764" height="580"></object>
```

The control has a default width and height of 764 x 580 so this should be specified unless the *PreviewAreaWidth* or *PreviewAreaHeight* properties are specifically set.

The object tag with the class ID is sufficient to run the control on a computer where the installer has been used. To run the control on another machine it will be necessary to add a CODEBASE attribute pointing to the CAB file. A signed CAB file is provided and this will need to be moved from the folder where it was installed to a folder within your web site. The CODEBASE attribute uses a relative path.

Example:

```
<object id="upload" classid="CLSID:078B03B7-23A0-4EA6-A0A9-6E27EA68BE79" codebase="csXThumbUpload.cab" width="764" height="580"></object>
```

1.1.1 Licensing and LPK Files

csXThumbUpload is a licensed control and so a Licence Package File is required on the server. This allows the licensed copy of the control to be placed on the server but an unlimited number of users can connect to the server and use the control without requiring their own licence.

A Licence Package File is supplied with the installer and it will be in the same folder as the csXThumbUpload control. This can be copied to the server, or a .lpk file can be generated on the server as described below.

A Licence Package File can be produced using the Microsoft LPK File Generation Tool, which can be downloaded - [here](#). Before running the tool, make sure the csXThumbUpload control has been registered on the server and the licence file (csXThumbUpload.lic or csXThumbUploadTrial.lic) is present in the same folder. Running the installer will achieve this. The LPK Tool will display a list of available controls and then prompt for a name and location of the .lpk file. Save this file into the folder in the web site where it will be used.

The .lpk file is called using the following code, which must be placed before the object tag which calls the csXThumbUpload control.

```
<object classid="clsid:5220cb21-c88d-11cf-b347-00aa00a28331">  
  <param name="LPKPath" value="csxthumbupload.lpk">  
</object>
```

The class ID is the same for all .lpk files. The value of the LPKPath parameter is a relative path pointing to the .lpk file and it must not start with "http://". The example above would be used unchanged if the .lpk file is in the same folder on the web server as the web page. The .lpk file supplied with the trial version of csXThumbUpload is called "csxthumbuploadtrial.lpk".

1.1.2 Parameters

The properties for csXThumbUpload can be set using PARAM tags, which are included inside the object tag which calls the control. Here is an example using the class ID for the full version:

```
<object id="upload" classid="CLSID:078B03B7-23A0-4EA6-A0A9-6E27EA68BE79"
width="764" height="580">
    <param name="RemoteURL" value="http://mysite.com/savefile.asp">
    <param name="ImagesOnly" value="True">
</OBJECT>
```

These tags would set the *RemoteURL* property, which is the script that will receive the uploaded files, as well as restricting the file types to supported images with the *ImagesOnly* property. The properties can also be set from a Javascript function.

1.1.3 Version

If a more recent version of the csXThumbUpload control is used in a web application it is necessary to create a new .lpk file using this version. The complete version and build number can be specified in the CODEBASE attribute and this will force the client browser to download the CAB file again if it is using an earlier version. The syntax is:

```
CODEBASE="csxthumbupload.cab#version=1,0,0,0"
```

To find the exact version number, right click on the OCX file and select "Properties". Do not right click on the .CAB file or the executable installer. At the time of writing this there is no newer version of csXThumbUpload available.

1.2 Using csXThumbUpload in Visual Basic

The csXThumbUpload control can be imported into the Tool Box through the Project menu by selecting Components. If it has been installed with our installer it will appear in the list of available controls. Tick the box and select Apply, then Close, and the control will now appear in the Tool Box and it can be dragged onto a form.

The *RemoteURL* property must always be set so that the control has a destination for HTTP uploads. It is possible to have an application for uploading that contains no other code. Alternatively for FTP uploads the *HTTPUpload* property would be set to false and the *FTPHost* property set to the FTP server host name.

1.3 Other Development Environments

csXThumbUpload is an ActiveX control so it can run in a wide range of COM enabled environments. You will need to refer to the documentation for your development environment for details of how to use ActiveX controls.

1.4 The Trial Version

The trial version of csXThumbUpload is supplied as a different OCX file from the full version, called csXThumbUploadTrial.ocx, and it has a separate installer. The trial version contains a limitation where a message box pops up after each file is uploaded. This message box is removed in the full version.

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

1.5 Supported Image Formats

Image files that are in a supported format can be previewed and edited. The supported formats are:

BMP, JPG, GIF, PNG, TIF, PSD and PCX.

They must have one of the above extensions although the variations .jpeg and .jpe (for .jpg) and .tiff (for .tif) are all accepted.

2 Properties Controlling Appearance and Layout

The overall width and height of the control will depend on the size of the preview area, which can be adjusted by setting the *PreviewAreaHeight* and *PreviewAreaWidth* properties.

Width	-	Integer. The overall width of the control. This is equal to <i>PreviewAreaWidth</i> + 216.
Height	-	Integer. The overall height of the control. This is equal to <i>PreviewAreaHeight</i> + 180.

The *Width* and *Height* properties are specified as attributes in the object tag when the control is called on a web page. They are not set like other parameters. By default they should be set to 764 and 580 for the *Width* and *Height* respectively.

PreviewAreaWidth	-	Integer. The width of the large rectangular area where the files are shown. This must have a value between 300 and 800. (Default = 548)
PreviewAreaHeight	-	Integer. The height of the large rectangular area where the files are shown. This must have a value between 300 and 600. (Default = 400)

The file previews are displayed on tiles which can have their size adjusted, as can the padding between them and the number of tiles in a row.

TileWidth	-	Integer. The width of each file preview tile, in pixels. This must have a value between 80 and 300. (Default = 120)
TileHeight	-	Integer. The height of each file preview tile, in pixels. This must have a value between 80 and 300. (Default = 120)
TilePadding	-	Integer. The space between each file preview tile, in pixels. This can have a maximum value of 50. (Default = 10)
TilesPerRow	-	Integer. The number of preview tiles shown in each row. This must have a value between 2 and 12. (Default = 4)

The colour can be specified for some elements of the control and these are specified as OLE_COLOR values. These are 32 bit integer values where the first byte is a flag and the remaining three are the blue, green and red components. For a specific RGB colour, the first byte is zero.

For example, blue is "&H00FF0000", green is "&H0000FF00" and red is "&H000000FF". To specify a system colour the first byte is hexadecimal 80. The *FormColor* and *TileColor* properties default to the system colour "&H8000000F" which is the colour for a button face.

FormColor	-	OLE_COLOR. The colour of the face of the control. (Default = &H8000000F, the colour of a button face.)
FormTextColor	-	OLE_COLOR. The colour of the text in the bottom section of the control which display the details of files selected and upload progress. (Default = 0 or black.)
FileNameColor	-	OLE_COLOR. The colour of the text showing each file name. (Default = 0 or black.)
PreviewAreaColor	-	OLE_COLOR. The colour of the background of the preview area. (Default = &H00FFFFFF or white).
TileColor	-	OLE_COLOR. The colour of the background of each file preview tile. (Default = &H8000000F, the colour of a button face.)

There is only limited control over colours. There are no properties to control the colours of the buttons, scrollbars or the folder view.

DisplayThumbnails - Boolean. When true, images in the selected folder will be displayed as a thumbnail if they are in a supported format. Other files will be displayed as icons. When this property is false, all the files will be displayed as icons to indicate the file type. (Default = false)

DisplayIconsFirst - Boolean. When *DisplayThumbnails* is true, setting *DisplayIconsFirst* to true will display all the files in the folder as icons before displaying any image thumbnails. This allows the user to begin selecting files for upload even if the thumbnail listing is incomplete. (Default = true)

MaxFileCount - Integer. Limits the maximum number of files that will be displayed. A value of zero will list all the files in the folder. The files are shown in alphabetical order so any limit will display files that are earlier in the alphabet. (Default = 0)

MaxFileSize - Integer. Limits the maximum file size, in bytes, of listed files. A value of zero will list all files regardless of size. (Default = 0)

FilterAdd (*Value As string*) - Integer return value. Method to add a file extension for use in filtering the file listing. *Value* is a file extension with or without the period character and it is not case sensitive. The return value is the number of filters added to the list so far. Wild cards are not permitted.

FilterByInclusion - Boolean. This determines if the files with the specified extensions are listed (true) or if they are excluded from the listing (false). (Default = true)

FilterClear - No return value. Clears any filters added by *FilterAdd*.

FilterAdd and *FilterClear* cannot be called with the <param> tag. In a web application they must be called from a Javascript function, for example a function called by the OnLoad event.

ImagesOnly - Boolean. Setting this to true is the equivalent of adding all the supported image file extensions with *FilterAdd* and setting *FilterByInclusion* to false. It should only be set as an alternative to adding individual filters. (Default = false)

3 Properties Controlling Uploads

Files can be uploaded either as an HTTP post or by FTP. The upload method is defined by the *HTTPUpload* Boolean property which is true for an HTTP upload and false for FTP. An HTTP upload can include form variables, added using the *FormVariableAdd*, method and it can optionally allow authentication using either Basic Authentication or Integrated Windows Authentication. An HTTP upload must have some sort of server side script to save the files.

FTP uploads cannot include any form variables. It can use either active mode or passive mode.

HTTPUpload - Boolean. When true the uploads will use HTTP and when false they will use FTP. (Default = true)

RemoteURL - String. The destination URL for an HTTP upload. This must include the "http://" or "https://" prefix. It must not be a relative path. (Default = empty string)

FormVariableAdd(*Name As String, Value As String*) - Integer. Add a form variable to send with the HTTP post. It takes a name and a value as string parameters. The return value is the number of variables added so far.

FormVariablesClear - Clears any form variables added with *FormVariableAdd*.

FormVariableAdd and *FormVariablesClear* cannot be called with the <param> tag. In a web application they must be called from a Javascript function, for example a function called by the OnLoad event.

FormTagName - String. The name of the form variable for the upload. This would be the name attribute if the upload used the input tag in a web page. Not all server side file saving components use it but it must be present. (Default = "csXThumbUpload")

AuthenticationType - Integer, 0 or 1 only. If HTTP Authentication is used on the receiving server this property specifies the type. Set to 0 for no authentication or Basic Authentication and 1 for Integrated Authentication. (Default = 0)

HTTPPassword - String. The password for HTTP authentication. If empty, a dialogue box will be displayed for the user to enter their own password and username. (Default = empty string)

HTTPUsername - String. The username for HTTP authentication. (Default = empty string)

Two properties provide information for debugging or monitoring status. *UploadStatus* is simply an integer value indicating a successful upload or an error. *ReturnText* is the full page returned by the remote script in an HTTP upload.

UploadStatus - Integer, read only. This indicates whether the last file was uploaded successfully. If the upload failed it will indicate the error. Possible values are:

- 0 - HTTP password dialogue cancelled by the user.
- 1 - HTTP authentication error.
- 2 - Server error during HTTP upload.
- 3 - Connection error during HTTP upload.
- 4 - FTP password dialogue cancelled by the user.
- 5 - Error setting remote folder during FTP upload.
- 6 - Server error during FTP upload.
- 7 - Connection error during FTP upload.
- 8 - No outgoing connection found.
- 9 - Invalid image that could not be loaded during editing.
- 10 - User abort.

It returns the HTTP code when a connection to the server was made and will be 200 for a completed upload. In the case of a successful FTP upload *UploadStatus* will also return 200.

ReturnText - String, read only. This contains the content returned by the remote script after uploading a file. It can be useful for debugging the remote script.

The following properties apply to FTP uploads. The username and password can be provided by setting *FTPUsername* and *FTPPassword* or these can be left blank and a dialogue box will ask the user for them.

FTPHost - String. The host computer used for an FTP upload. This may take the form "ftp.somehost.com" or it may be an IP address. (Default = empty string)

FTPPassiveMode - Boolean. Determines if passive mode (true) or active mode (false) is used. (Default = true)

FTPPassword - String. The FTP password. If left empty a dialogue box will be displayed for the user to enter their own password and username. (Default = empty string)

FTPUsername - String. The FTP username. (Default = empty string)

FTPPort - Integer. The port used by the FTP server. (Default = 21)

FTPRemoteFolder - String. A sub folder or path on the FTP server where files will be saved. (Default = empty string)

HideErrors - Boolean. When true, pop up error messages will be disabled for errors during uploading. Errors can be identified using the *OnError* event to provide customised messages. (Default = false)

TimeOut - Integer. The length of inactivity in seconds before a connection will time out. (Default = 60)

4 Properties Controlling Image Editing

An important feature of csXThumbUpload is the ability to perform some image edits on images that are in a supported format, before uploading the image. The edits will apply to all the images uploaded at the same time and the instructions are given by setting properties.

Edits include resizing, changing image format, changing colour depth, or removing metadata.

ApplyEdits - Boolean. When true any image uploaded will be loaded into the control and then uploaded. Any specified edits will be carried out during the process. (Default = false)
--

It is important to be aware that when *ApplyEdits* is false, files are uploaded unedited. When *ApplyEdits* is true, only images that are successfully loaded by csXThumbUpload will be uploaded and they will probably be changed in some way (such as the compression of a JPG image) even if no edits have been defined.

ResizeImage - Boolean. When true the selected images will be resized before uploading. The new size is given by <i>NewWidth</i> and <i>NewHeight</i> . If either parameter is zero the other will be used to determine the size, and aspect ratio will be maintained. (Default = false)
--

NewHeight - Integer. New image height when <i>ResizeImage</i> is true. (Default = 0)

NewWidth - Integer. New image width when <i>ResizeImage</i> is true. (Default = 0)

ResizeFit - Boolean. When true the selected images will be resized before uploading. The new size is given by a maximum height, <i>MaxHeight</i> , and a maximum width, <i>MaxWidth</i> . Aspect ratio will always be maintained, and if the original image is already within the specified size, it will be unchanged. (Default = false)
--

MaxHeight - Integer. New maximum image height when <i>ResizeFit</i> is true. (Default = 0)

MaxWidth - Integer. New maximum image width when <i>ResizeFit</i> is true. (Default = 0)

ClearICCProfile - Boolean. When JPG and TIF images contain an embedded ICC colour profile this will be preserved by default during the upload and edit. To remove any embedded colour profile, set <i>ClearICCProfile</i> to true. (Default = false)

JPG, TIF and PSD images can contain metadata and some of this will be retained by csXThumbUpload when these images are edited. The supported metadata properties fall into three categories, each having its own section within the file. These are IPTC (IIM specification), Exif and XMP. Default behaviour is to retain metadata.

ClearMetaData - Boolean. When true, all the metadata in the image will be removed. (Default = false)

KeepExif - Boolean. If <i>ClearMetaData</i> is true, <i>KeepExif</i> can be set to true to retain the Exif data block. (Default = false)

KeepIPTC - Boolean. If <i>ClearMetaData</i> is true, <i>KeepIPTC</i> can be set to true to retain the IPTC data block. (Default = false)

KeepXMP - Boolean. If <i>ClearMetaData</i> is true, <i>KeepXMP</i> can be set to true to retain the XMP data block. (Default = false)
--

NewColorDepth - Integer, 0, 1, 4, 8 or 24 only. The colour depth of an image can be changed during upload by setting the <i>NewColorDepth</i> property. Supported values are 1, 4, 8 and 24 for 1-bit, 4-bit, 8-bit and 24-bit images. A value of zero leaves the colour depth unchanged. (Default = 0)
--

NewFormat - Integer, 0 to 7 only. This can change the format of the uploaded image to a new format. Possible values are:

0 - Unchanged. (Default)

1 - BMP.

- | | | |
|---|---|------|
| 2 | - | JPG. |
| 3 | - | GIF. |
| 4 | - | PNG. |
| 5 | - | TIF. |
| 6 | - | PSD. |
| 7 | - | PCX. |

GrayScale - Boolean. When true, the image will be converted to 8 bit greyscale. If the image is a JPG it will also be converted to greyscale. (Default = false)

CompressionType - Integer, 0 to 3 only. When TIF images are saved they can be compressed using a choice of methods by setting this property. Possible values are:

- | | | |
|---|---|----------------------|
| 0 | - | Unchanged. (Default) |
| 1 | - | PackBits. |
| 2 | - | Group 4. |
| 3 | - | None. |

Leaving the value at zero, for unchanged, will use the same compression method as the original TIF. If it uses LZW compression it will be converted to PackBits, and if it uses Group 3 it will be converted to Group 4, because these compression types are not supported for writing.

<p>JPGQuality - Integer, 1 to 100. This is a measure of the amount of compression used when saving a JPG image. A high value is a higher quality higher file size while a lower value is a lower quality and a smaller file size. Adjusting JPG compression is an easy way to reduce the file size of an image but setting a value for <i>JPGQuality</i> that is too low will result in a poor quality image. It is the differences in compression / quality that lead to large changes in file size when an image is saved without performing any other edits. (Default = 90)</p>

5 Properties Controlling Displayed Text

The button and status labels and main error messages are string properties which can be set. This may be useful for translating the control for use in other languages.

Text_SelectButton	-	String. The text on the Select All button. (Default = "Select All")
Text_UnselectButton	-	String. The text on the Unselect All button. (Default = "Unselect All")
Text_UploadButton	-	String. The text on the Upload button. (Default = "Upload")
Text_CancelButton	-	String. The text on the Cancel button. (Default = "Cancel")

The buttons are a fixed size so any text used for the labels should be of a comparable length to the default text.

Text_FilesSelected	-	String. The text introducing the number of files selected. (Default = "Files Selected")
Text_Uploading	-	String. The text introducing the name of the file currently uploading. (Default = "Uploading")
Text_Waiting	-	String. The text which indicates the control is waiting for a response from the server after uploading. (Default = "...waiting for reply from server.")
Text_Complete	-	String. The text which displays briefly to indicate a successful upload. (Default = "Complete.")
Message_ConnError	-	String. Pop up error message when connection to the server fails. (Default = "Error connecting to the server.")
Message_AuthError	-	String. Pop up error message indicating authentication failure. (Default = "Error with server authentication.")
Message_UploadError	-	String. Pop up error message for an unspecified upload failure. The file name will be displayed after this string. (Default = "Error uploading file:")
Message_FolderError	-	String. Pop up error message when failing to set a remote folder by FTP. (Default = "Error setting remote folder.")
Message_InternetError	-	String. Pop up message warning that no outgoing connection could be found. (Default = "Error finding a connection.")
Message_ImageError	-	String. Pop up error message warning of an invalid image during editing. It is preceded by the file name. (Default = "is not a valid image.")
TextPass_Caption	-	String. The caption on the password form. (Default = "Enter Network Password")
TextPass_Note	-	String. The descriptive note on the password form. (Default = "Please type your user name and password.")
TextPass_Site	-	String. The label introducing the site or domain name on the password form. (Default = "Site")
TextPass_Username	-	String. The label introducing the user name on the password form. (Default = "User Name")
TextPass_Password	-	String. The label introducing the password on the password form. (Default = "Password")

6 Events

csXThumbUpload supports a number of standard events, OnActivate, OnClick, OnCreate, OnDbClick, OnDestroy, OnDeactivate, OnKeyPress and OnPaint. In addition to these there are six events that are triggered at various stages within the upload process.

<p>OnUploadStart - Triggered when the upload process begins. It could be useful for adding form variables that have been collected from another part of the application.</p> <p>OnUploadComplete - Triggered when an upload batch process ends, including when the files have all been uploaded, when an error occurs and when the user aborts. The <i>UploadStatus</i> property should be checked to verify why the process stopped. For example, <i>UploadStatus</i> will be 200 after successful completion and 10 after a user abort.</p> <p>OnFileStart(<i>FileNumber</i> As Integer, <i>FileTotal</i> As Integer) - Triggered as each file starts to upload. <i>FileNumber</i> counts which file is being uploaded starting with one for the first file in the list. <i>FileTotal</i> is the total number of files selected for upload.</p> <p>OnFileComplete(<i>FileNumber</i> As Integer, <i>FileTotal</i> As Integer) - Triggered after each successful file upload. <i>FileNumber</i>, and <i>FileTotal</i> are as described above.</p> <p>OnAbort - Triggered when the user presses the Cancel button to abort the upload.</p> <p>OnError(<i>FileNumber</i> As Integer, <i>FileTotal</i> As Integer, <i>Code</i> As integer) - Triggered when an error occurs during an upload. <i>FileNumber</i> and <i>FileTotal</i> are as described above. <i>Code</i> is an integer value indicating where the error occurred. <i>Code</i> will be 0 to 9 and has the same values as <i>UploadStatus</i>.</p>

The way in which events are handled varies depending on the programming language. Here is an example of using the *OnFileComplete* event in Javascript to display the value of the *ReturnText* property. This would usually be used for debugging because it displays the contents of the server side script. There are other occasions where it might be useful to display a message from the server side script after an upload.

```
<object id="upload" classid="CLSID:078B03B7-23A0-4EA6-A0A9-6E27EA68BE79"
width="764" height="580">
<param name="RemoteURL" value="http://mysite.com/savefile.asp">
</object>
<script language="javascript">
function DisplayReturnText()
{
x=document.getElementById("returnfile");
x.innerHTML = upload.ReturnText;
y = document.getElementById("uploadstatus");
y.innerHTML = upload.uploadstatus;
}
</script>

<script language="javascript" for="upload" event="OnFileComplete(FileNumber,
FileTotal)">
DisplayReturnText();
</script>

<div id="returnfile"></div>
<div id="uploadstatus"></div>
```

The event handler is placed inside a `<script>` tag where the FOR attribute matches the ID attribute for the object and the EVENT attribute specifies the event. The code inside the tag will execute after each file has been uploaded. The variables *FileNumber* and *FileTotal* are defined in the event handler but they are not used in this example.

In this example the text returned by the server side file saving script is written to one <div> tag while the value of the *UploadStatus* property is written to another.

The *csXThumbUpload* object cannot be used directly inside the event handlers so the Javascript function *DisplayReturnText* has been defined and the code placed there.

We have shown the *OnFileComplete* event because it has parameters. If this is to be used in debugging an error it should be used with the *OnUploadComplete* event because *OnFileComplete* is only triggered after a successful upload.

7 Code Examples

Here are some examples showing how properties are set to change the behaviour of csXThumbUpload.

Example 1 - Resizing Images

The following code uses the full version of csXThumbUpload to select images, display them as thumbnails and then resize them during upload.

```
<object classid="clsid:5220cb21-c88d-11cf-b347-00aa00a28331">
<param name="LPKPath" value="csxthumbupload.lpk" />
</object>
<object id="upload" classid="CLSID:078B03B7-23A0-4EA6-A0A9-6E27EA68BE79"
width="764" height="580" codebase="csXThumbUpload.cab">
<param name="RemoteURL"
value="http://mysite.com/uploads/save.asp" />
<param name="HTTPUpload" value="true" />
<param name="ImagesOnly" value="true" />
<param name="DisplayThumbnails" value="true" />
<param name="DisplayIconsFirst" value="true" />
<param name="ApplyEdits" value="true" />
<param name="ResizeFit" value="true" />
<param name="MaxWidth" value="200" />
<param name="MaxHeight" value="200" />
</object>
```

The first object tags loads the licence package file, as described in section 1.1.1. The second object tag loads the full version of csXThumbUpload. For the trial version change this class ID as described in Section 1.1 and also change the names of the two files which are used, the .lpk and .cab files.

The *RemoteURL* property must always be set for an HTTP upload and this is the URL to the script on the server that will save the upload. It must not be a relative path and must always start with "http://" or "https://".

The next three properties control what is displayed by the control. They will be supported images and they will be displayed as thumbnails, but they will be listed as icons first. The thumbnail preview takes a noticeable amount of time and the user can begin selecting files once the icon listing is shown.

Finally, *ApplyEdits* and *ResizeFit* are set to true so the images will be resized before uploading and the maximum overall dimensions will be given by *MaxWidth* and *MaxHeight*. This type of resize will always maintain the aspect ratio.

Example 2 - FTP Upload

The following code shows the parameter settings to upload files by FTP. The object tags are not shown this time.

```
<param name="HTTPUpload" value="false" />
<param name="FTPHost" value="ftp.mysite.com" />
```

Files are uploaded by FTP when the HTTPUpload property is set to false. The FTPHost property is set to the host address of the FTP server. If the username and password are not set as parameters the user will be prompted to enter them. Once they have authenticated they will not be asked again until the control is reloaded. The FTP session will only last until the selected files have been uploaded, but the username and password will be remembered for the next upload.

Example 3 - Setting Filters and Form Variables

Files can be filtered by extension but the filters are added using the method FilterAdd. This cannot be set as a parameter inside the object tag so it must be called from inside a Javascript function. The code shown below is the Javascript function to add a filter to display files with a ".jpg" extension only. It also

adds a form variable to be passed with an HTTP upload. The Javascript code would be placed inside the HTML <head> area.

```
<script language="javascript">
  function Setup()
  {
    upload.FilterAdd("jpg");
    upload.FormVariableAdd("VariableName", "Value");
  }
</script>
```

The csXThumbUpload object has the ID "upload" as in Example 1, above.

The Setup function can be called in the onload event for the page, using the following body tag.

```
<body onload="Setup();">
```

It is also possible to set the properties inside a Javascript function like this instead of using the param tag.

8 Saving Files Uploaded by HTTP

The script that receives the uploads on the remote server needs to be able to save the files. A method of doing this is described briefly here for four of the main scripting environments.

It is important to remember that each file is sent as a separate upload so if there are 20 files selected for upload the script will be called 20 times. The permissions might need to be configured to allow the script to save the files. In IIS in Windows the scripts run as the Internet Guest User and this account must have Write permission on the server. All four examples shown here are simply saving each file into the same folder as the script and keeping the name the same.

The content of the remote script is written to the *ReturnText* property after uploading and this can be read to assist in debugging. The *ReturnText* property could be read from inside the *OnUploadComplete* event handler as this is called even when there is an error with the upload, and the value could be displayed in a message box such as the alert box in Javascript, or written to another part of the web page.

csXThumbUpload supports server authentication. The *AuthenticationType* property must be set to 1 if Windows Integrated Authentication (NTLM) is used, otherwise the uploads will fail. With Basic Authentication or no authentication, *AuthenticationType* must be set to 0. As this is the default value, the property can be left unused. The user name and password can be coded into the application by setting the *HTTPUsername* and *HTTPassword* properties. If these properties are not set, a dialogue box will be used to collect the log in details from the user.

8.1 ASP

There is no built in method for saving an HTTP upload in ASP and a third party component is required. We supply an ASP component with csXThumbUpload which can save single uploads and also read form variables. It is called ASPCustomUpload and it has its own documentation but a simple script is shown below which saves the file into the same folder as the script. In practice a separate folder should be used for saving uploads. If using a different component refer to the documentation for that component.

```
<%@ language=vbscript %>
<%
Set Upload = Server.CreateObject("ASPCustomUpload.FileSave")
Upload.SaveFile Server.MapPath(Upload.FileName)
%>
```

The ASPCustomUpload component must be registered on the web server before running this script.

The *FormTagName* property of csXThumbUpload is not needed with this component, but some other components may need the value of this property when saving the file. It is set to "csXThumbUpload" by default.

8.2 ASP.NET

The *HTMLInputFile.PostedFile* method of saving an uploaded file in ASP.NET cannot be used with csXThumbUpload. We supply a .NET class with csXThumbUpload which can save single uploads. It is called NetCustomUpload and it has its own documentation but a simple script is shown which saves the file into the same folder as the script.

```
<%@ Page language="vb" %>
<%@ Import Namespace = "NetCustomUpload" %>
<%
Dim Upload As New UploadClass
Upload.ReadUpload
```

```
Upload.SaveFile(Server.MapPath("./files/") & Upload.Filename)
%>
```

The file Chestysoft.NetCustomUpload.dll must be placed in the bin folder for the web application before running this script.

The *FormTagName* property of csXThumbUpload is not needed with this component, but some other components may need the value of this property when saving the file.

8.3 Cold Fusion

In Cold Fusion the cfile tag saves uploaded files. It has attributes for renaming files to avoid name conflicts. Here is a simple script that saves the files into the same folder and uses the original names.

```
<cfset CurrentDir = ExpandPath(".")>
<cfset filename=CurrentDir & "\">
<cfile action="upload" filefield="csXThumbUpload" destination=#filename#>
```

The *FormTagName* property of csXThumbUpload is required for the filefield attribute. The default value of "csXThumbUpload" has been used.

8.4 PHP

PHP can also save the file directly. Here is a simple script that saves into the same folder and it keeps the original file name.

```
<?php
    move_uploaded_file($_FILES['csXThumbUpload']['tmp_name'],
$_FILES['csXThumbUpload']['name'])
?>
```

The *FormTagName* property of csXThumbUpload is required to identify the file for saving. The default value of "csXThumbUpload" has been used.

9 Other Products From Chestysoft

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

ActiveX Controls

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

- [csImageFile](#) - ASP component to resize and edit images.
- [csDrawGraph](#) - Component to draw pie charts, bar charts and line graphs.
- [csIniFile](#) - Read and Edit Windows style inifiles.
- [csASPUpload](#) - Process file uploads through a browser.
- [csASPZipFile](#) - Create zip files and control binary file downloads.
- [csFileDownload](#) - Control file downloads with an ASP script.
- [csASPGif](#) - Create and edit animated GIFs.
- [csFTPQuick](#) - ASP component to transfer files using FTP.

ASP.NET

- [csASPNetGraph](#) - A .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.](#)

10 Alphabetical Index of Commands

Command	Page no.	Command	Page no.
ApplyEdits	10	NewColorDepth	10
AuthenticationType	8	NewFormat	10
ClearICCProfile	10	NewHeight	10
ClearMetaData	10	NewWidth	10
CompressionType	11	OnAbort	13
DisplayIconsFirst	7	OnError	13
DisplayThumbnails	7	OnFileComplete	13
FileNameColor	6	OnFileStart	13
FilterAdd	7	OnUploadComplete	13
FilterByInclusion	7	OnUploadStart	13
FilterClear	7	PreviewAreaColor	6
FormColor	6	PreviewAreaHeight	6
FormTagName	8	PreviewAreaWidth	6
FormTextColor	6	RemoteURL	8
FormVariableAdd	8	ResizeFit	10
FormVariablesClear	8	ResizeImage	10
FTPHost	9	ReturnText	9
FTPPassiveMode	9	Text_CancelButton	12
FTPPassword	9	Text_Complete	12
FTPPort	9	Text_FilesSelected	12
FTPRemoteFolder	9	Text_SelectButton	12
FTPUsername	9	Text_UnselectButton	12
GrayScale	11	Text_UploadButton	12
Height	6	Text_Uploading	12
HideErrors	9	Text_Waiting	12
HTTPPassword	8	TextPass_Caption	12
HTTPUpload	8	TextPass_Note	12
HTTPUsername	8	TextPass_Password	12
ImagesOnly	7	TextPass_Site	12
JPGQuality	11	TextPass_Username	12
KeepExif	10	TileColor	6
KeepIPTC	10	TileHeight	6
KeepXMP	10	TilePadding	6
MaxFileCount	7	TilesPerRow	6
MaxFileSize	7	TileWidth	6
MaxHeight	10	TimeOut	9
MaxWidth	10	UploadStatus	8
Message_AuthError	12	Width	6
Message_ConnError	12		
Message_InternetError	12		
Message_UploadError	12		
Message_FolderError	12		
Message_ImageError	12		

