



Website: www.chestysoft.com

Email: info@chestysoft.com

csXPostUpload - Version 1.1

ActiveX Control to Post Files to a Server Using HTTP

This ActiveX control allows the end user to upload files from a folder to a remote server. It uses an HTTP post operation to send the files in the same way that a browser form can upload files using the form tag `<input type=file...>`. The server must be running a script or cgi that is capable of saving the files.

csXPostUpload 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 files to be uploaded are selected by specifying the source folder, and some filtering can be applied to add or remove specific file types, and to restrict size. When the Upload button is clicked each file is posted to the URL in turn, complete with any form variables. There is a separate post operation for each file.

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](#)
- [The User Interface](#)
- [Saving Uploads on a Server](#)
- [Methods, Properties and Events](#)
- [Hiding the Interface and Selecting Files Programmatically](#)
- [Deploying an Application](#)
- [Alphabetical List of Commands](#)

TABLE OF CONTENTS

1. INSTALLATION AND GETTING STARTED	3
1.1. USING CSXPOSTUPLOAD IN INTERNET EXPLORER	3
1.1.1. Licensing and LPK Files.....	3
1.1.2. Parameters.....	4
1.2. USING CSXPOSTUPLOAD IN VISUAL BASIC	4
1.3. OTHER DEVELOPMENT ENVIRONMENTS	5
1.4. THE TRIAL VERSION.....	5
2. METHODS, PROPERTIES AND EVENTS	6
2.1. METHODS	6
2.2. PROPERTIES	6
2.3. EVENTS.....	8
3. THE USER INTERFACE.....	10
4. IMPORTANT POINTS.....	11
4.1. THE FORMTAGNAME PROPERTY	11
4.2. FORM VARIABLES.....	11
4.3. MOVING AND DELETING LOCAL FILES	11
4.4. FILTERS	12
4.5. UPLOAD, ABORT, CONTINUE AND RESET	12
4.6. STORED SETTINGS - CSXPOSTUPLOAD.INI.....	13
4.7. SERVER AUTHENTICATION	13
5. HIDING THE INTERFACE AND SELECTING FILES BY NAME.....	14
6. SAVING UPLOADS ON A SERVER.....	15
6.1. ASP.....	15
6.2. ASP.NET	15
6.3. COLD FUSION.....	16
6.4. PHP.....	16
7. DEPLOYING AN APPLICATION.....	17
7.1. COMPILED APPLICATIONS (E.G VISUAL BASIC OR DELPHI)	17
7.2. WEB BROWSER APPLICATIONS (E.G. JAVASCRIPT)	17
8. OTHER PRODUCTS FROM CHESTYSOFT	18
9. ALPHABETICAL LIST OF COMMANDS.....	19

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.

1.1. Using csXPostUpload in Internet Explorer

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

Full Version

```
<object id="upload" classid="CLSID:E309D65D-25E3-4E75-8AC2-BBA044724875" width="700" height="400" border="1"></object>
```

Trial Version

```
<object id="upload" classid="CLSID:619543FE-E15A-45D4-9D96-28F33E79AB59" width="700" height="400" border="1"></object>
```

The width and height are always 700 and 400 respectively and if smaller values are used part of the control will be hidden and it could be unusable. The border attribute is optional.

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 OCX file or a CAB file. The CODEBASE attribute uses a relative path.

Example:

```
<object id="upload" classid="CLSID:E309D65D-25E3-4E75-8AC2-BBA044724875" width="700" height="400" border="1" codebase="csXPostUpload.ocx" ></object>
```

1.1.1. Licensing and LPK Files

csXPostUpload 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 csXPostUpload 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 csXPostUpload control has been registered on the server and the licence file (csXPostUpload.lic or csXPostUploadTrial.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 csXPostUpload control.

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

The class ID is the same for all .lpk files. The LPKPath parameter is a relative path pointing to the .lpk file. A full URL beginning with "http://" must not be used for this parameter.

1.1.2. Parameters

The properties that control the display can be added as "PARAM" tags, or in a Javascript function that can be called using the *OnLoad* event of the page. Methods that add form variables or filters must be called from a Javascript function.

Example of PARAM tags:

```
<object id="upload" classid="CLSID:E309D65D-25E3-4E75-8AC2-
BBA044724875" width="700" height="400" border="1">
<param name="RemoteURL" value="http://mysite.com/savefile.asp">
<param name="EnableURL" value="false">
<param name="FormTagName" value="fileupload">
</object>
```

This sets the URL and also disables the URL edit box so that it cannot be changed by the user. It also specifies the *FormTagName* property which is used by most file upload scripts and components when the file is saved. The example above is the equivalent of the following form tag `<input type="file" name="fileupload">`.

Example of a Javascript onLoad function:

```
<script language="JavaScript">
<!--
    function Setup()
    {
        upload.AddVariable('BatchName', '');
    }
//-->
</script>
<body onload="Setup()">
<object id="upload" classid="CLSID:E309D65D-25E3-4E75-8AC2-
BBA044724875" width="700" height="400" border="1"></object>
```

Here a Javascript function called Setup is run when the page loads. This calls the *AddVariable* method which adds a form variable called BatchName with a value to be entered by the user. Properties could also be set in this function instead of using the PARAM tag.

1.2. Using csXPostUpload in Visual Basic

The csXPostUpload control is 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 controls. Tick the box, select Apply, then Close and the control will now be in the Tool Box and it can be dragged onto a form.

The control should be 700 pixels wide and 400 pixels high. It can also be set to the correct size by setting *AutoSize* to true in the Properties Window. Most of the relevant properties that determine the appearance of the control are not available in the Properties Window and they need to be set from code. The *OnLoad* event of the parent form is a suitable place for setting these properties.

Example:

```
Private Sub Form_Load()
    Upload1.EnableURL = False
    Upload1.RemoteURL = "http://www.mysite.com/savefile.asp"
    Upload1.FormTagName = "fileupload"
End Sub
```

The csXPostUpload control has the default name of "Upload1". This example shows how to set the URL and also to disable the edit box so that the end user cannot change this URL. It specifies the FormTagName property which is used by most file upload scripts and components when the file is saved. The example above is the equivalent of the following form tag in an HTML page <input type="file" name="fileupload">.

csXPostUpload needs an internet connection in order to upload files. It does not dial a connection.

There are two downloadable VB demos available from the Chestysoft website.

<http://www.chestysoft.com/xpostupload/vbexample1.asp>

<http://www.chestysoft.com/xpostupload/vbexample2.asp>

1.3. Other Development Environments

csXPostUpload 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 csXPostUpload is supplied as a different OCX file, called csXPostUploadTrial.ocx, and has a separate installation programme. The trial is date limited and after the expiry date the Upload button remains disabled and no files can be uploaded. Apart from this it is fully functional and has no missing properties. The expiry date is shown in the About box.

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

2. Methods, Properties and Events

2.1. Methods

The following methods are available.

About()	-	Displays the About box for the control. In the trial version this shows the expiry date.
AddVariable(<i>Name</i> As String, <i>Value</i> As String)	-	Adds a form variable which will appear in the "Form Variables" box in the control. <i>Name</i> is the variable name and <i>Value</i> is the value, which can be an empty string if this is to be entered by the user.
AddHiddenVariable(<i>Name</i> As String, <i>Value</i> As String)	-	Adds a form variable which does not appear in the control.
ClearVariables()	-	Removes any variables added by <i>AddVariable</i> .
ClearHiddenVariables()	-	Removes any variables added by <i>AddHiddenVariable</i> .
AddFilter(<i>Extension</i> As String)	-	Adds <i>Extension</i> to the list of file extensions used as filters. <i>Extension</i> should not include the period character or wildcards. e.g "jpg" not "*.jpg".
ClearFilters()	-	Clears the list of extensions used as filters.
Upload()	-	Starts the upload programmatically instead of using the button.
Reset()	-	Resets the lists of files to be uploaded.
AddFile(<i>FileName</i> As String)	-	Adds a file to the list of files to be uploaded. Files added this way are not shown in the control and are only uploaded when <i>ManuallyAddFiles</i> is true.
ClearFiles()	-	Clears the list of files added with <i>AddFile</i> .

2.2. Properties

The following properties are available. The default values are shown in brackets.

SourceFolder	-	String. Local folder from which files will be uploaded. (empty)
ShowSourceFolder	-	Boolean. Show or hide the source folder edit box. (true)
EnableSourceFolder	-	Boolean. Enable or disable the source folder edit box. (true)
RemoteURL	-	String. Address of script to which files will be posted. this must include the "http://" or "https://" prefix. It cannot use any other protocol. (empty)
ShowURL	-	Boolean. Show or hide the URL edit box. (true)
EnableURL	-	Boolean. Enable or disable the URL edit box. (true)
LocalDestination	-	String. Folder to which files will be moved after uploading. If this property is not set the files will remain in the source folder. (empty)
ShowLocalDestination	-	Boolean. Show or hide the local destination edit box. (true)
EnableLocalDestination	-	Boolean. Enable or disable the local destination edit box. (true)
ShowFilterButton	-	Boolean. Show or hide the filter button that opens the filter dialogue box. (true)

ShowProgress	-	Boolean. Show or hide the overall progress bar. (true)
ShowFileProgress	-	Boolean. Show or hide the individual file progress bar. (true)
ShowVariables	-	Boolean. Show or hide the form variables. (true)
ShowFilesSelected	-	Boolean. Show or hide the files selected for upload in a list box. (true)
ShowFilesSent	-	Boolean. Show or hide the files after uploading in a list box. (true)
WarnNoVariables	-	Boolean. When true the upload will not start if all the form variable values are empty. (false)
FilterByInclusion	-	Boolean. When true the list of file extensions used for filtering define file types included for upload. When false, the extensions define files excluded from the upload. (true)
LimitFileSize	-	Boolean. When true the files listed for upload will have a maximum size defined by <i>MaxFileSize</i> . (false)
MaxFileSize	-	Integer. The maximum file size of files in KB selected for upload. When zero, there is no size limit. <i>LimitFileSize</i> must be true to take effect. (0)
DeleteFiles	-	Boolean. When true, files will be deleted from the source folder after uploading. This should be used with caution because there is no guarantee the files have been successfully saved after uploading. It is not recommended to delete the local files unless they are copies of files stored elsewhere. (false)
FormTagName	-	String. This is the name of the form variable containing the file itself. Many components that save files on the server use this to identify the file. (empty)
ManuallyAddFiles	-	Boolean. When true the files that are uploaded are taken from the list created by the <i>AddFile</i> method, not the files shown in the control. This allows files to be selected programmatically instead of by specifying a directory and filters. (false)
FileCount	-	Integer. The number of files in the list created by <i>AddFile</i> . Files are removed from the list as they are uploaded. Read Only.
ShowProgressOnly	-	Boolean. When true the visible part of the control will be reduced in size to show the progress bars only and the abort button. (false)
ShowProgressOnlyNoAbort	-	Boolean. When <i>ShowProgressOnly</i> is true this will also hide the abort button. (false)
ShowMessages	-	Boolean. Show or hide the warning messages that are caused by errors. This should be set to false if the control is hidden or run programmatically. (true)
UploadStatus	-	Integer. This read only property indicates whether the last file was uploaded successfully. It can take the following values:
-1		Undefined. The value before uploading starts.
0		The server refused the file.
1		No files selected for upload.
2		No form variables when <i>WarnNoVariables</i> is true.
3		Connection failed. Possibly an incorrect URL or no connection available.
4		User abort.
It returns the HTTP code when a connection to the server was made and will be 200 for a completed upload. It can be used when <i>ShowMessages</i> is false and when the upload is controlled programmatically.		
ReturnText	-	String. This read only property contains the content returned by the remote script after uploading a file. It can be useful for debugging the remote script.

ContinueOnError	-	Boolean. The default behaviour is to stop the upload completely if there is an error. Setting this property to true will allow the upload to continue by pressing the Continue button. If a particular file is causing the error it will be unable to continue. (false)
AuthenticationType	-	Integer, 0 or 1 only. When the server requires authentication this must be set to 0 for Basic Authentication and 1 for Windows Integrated (NTLM) Authentication. It must always be set to 0 when no authentication is used. (0)
Username	-	String. This is the user name that will be sent with the upload, if authentication is required.
Password	-	String. This is the password that will be sent with the upload, if authentication is required.

If authentication is used and the *Username* and *Password* properties are not set, a dialogue box will be used to collect the user name and password from the user.

Visible	-	Boolean. Show or hide the control. (true)
AutoSize	-	Boolean. When true the control is shown full size regardless of the specified size. Applicable only in visual programming environments. (false)

2.3. Events

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

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.
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 4 after a user abort.
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.
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.
OnAbort	-	Triggered when the user presses the Abort button.

The way in which events are handled varies depending on the programming language. Here is an example of using the *OnUploadStart* and *OnFileStart* events in Javascript to add some form variables.

```
<object id="upload" classid="CLSID:E309D65D-25E3-4E75-8AC2-
BBA044724875" width="700" height="400">
<param name="RemoteURL" value="http://mysite.com/savefile.asp">
</object>
<script language="JavaScript">
function AddVariable()
{
upload.ClearVariables();
upload.AddHiddenVariable("Name", "Value");
}
function AddLastVariable()
```

```
{
upload.AddHiddenVariable("LastFile", "true");
}
</script>
<script language="JavaScript" for="upload" event="onUploadStart()">
AddVariable();
</script>
<script language="JavaScript" for="upload"
event="onFileStart(FileNumber, FileTotal)">
if (FileNumber == FileTotal)
{
AddLastVariable();
}
</script>
```

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 when the upload process is started.

In this example a hidden form variable is added when the uploading process starts, then an extra form variable is added before the last file is sent so that the server side script can detect the last file in the batch.

The csXPostUpload object cannot be used directly inside the event handlers so the Javascript functions *AddVariable* and *AddLastVariable* have been defined to set the variables.

3. The User Interface

csXPostUpload is designed to assist in automating the upload of a batch of files, perhaps as part of a web based control panel, and to overcome the limitation of the browser "Browse" button that only allows single files to be selected. Most of the properties, such as the local folder, the remote URL and the form variables can be set by the end user or the developer. Many of the properties can be set by the developer and concealed from the end user in order to simplify the process. The user interface can even be hidden completely and files selected programmatically.

The screen shot below shows the control in its default state with all the fields visible but empty.

The screenshot shows a web-based control panel for file uploads. It features several input fields and a table for configuration. On the right side, there are two empty boxes for displaying file lists. At the bottom, there are four buttons for controlling the upload process.

Variable Name	Value

The files in the source folder will be listed in the box marked "Files selected for upload", subject to any filter conditions. Once some files are available the Upload button will be enabled. There is an option for specifying a local folder where the files will be moved to after uploading. If any form variables are to be sent with the upload their names must be specified by the developer in the calling application. They will then appear in the box marked "Form Variables". Progress bars appear at the bottom right to show progress of both the individual file and the entire batch. The upload can be aborted by the user and continued.

4. Important Points

The following sections describe a variety of important points.

4.1. The FormTagName Property

This property might need to be set, depending on the requirements of the script on the remote server that is saving the files. It is the equivalent of the name of the file tag in an HTML form:

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

It cannot be set by the end user of the control, and if required it must be set in the code of the calling application. It is required by many ASP components, PHP and Cold Fusion where the script needs this name to extract the file from the uploaded data. Our own ASP component, csASPUUpload does not need this property, although it can be used.

4.2. Form Variables

An HTTP post can include form variables as well as file data. csXPostUpload allows the developer to specify form variables that can have values entered by the end user. The variable names are added in the calling application by using the *AddVariable* function. The syntax is:

AddVariable(*Name*, *Value*) where *Name* and *Value* are strings.

The name and value will appear in the Form Variables box and the user can overwrite the value. *Value* can be an empty string or it can be given a value to appear in the control as a default.

A simple check can be made to see if the user has entered values for form variables. When the *WarnNoVariables* property is set to true an error message will pop up when the Upload button is clicked if all the form variable values are empty. There is no scope for checking individual variable values before uploading.

Variables can also be added to the post without having them displayed in the control. These are added using the *AddHiddenVariable* function, which has the syntax:

AddHiddenVariable(*Name*, *Value*) where *Name* and *Value* are strings.

The form variables can be cleared programmatically by calling the methods *ClearVariables* and *ClearHiddenVariables*.


The same variables and their values are sent with each file in the batch. It is possible to modify the hidden variables from inside the event handlers by calling *ClearHiddenVariables* and *AddHiddenVariables*. The *OnFileStart* event can identify the number of the file being uploaded and the total number of files so this information could be included with the file upload. Calling the *AddVariable* and *ClearVariables* methods has no effect once the upload process has started.

4.3. Moving and Deleting Local Files

There is a choice of what to do with the files in the source folder after they have been uploaded. If a valid folder is specified in the *LocalDestination* property, the files will be moved to this folder. They are moved, not copied, so there will not be a copy left in the source folder. If no valid folder is specified by the *LocalDestination* property the files remain in the source folder. There is nothing to prevent the user from clicking the Reset button and uploading them all again.

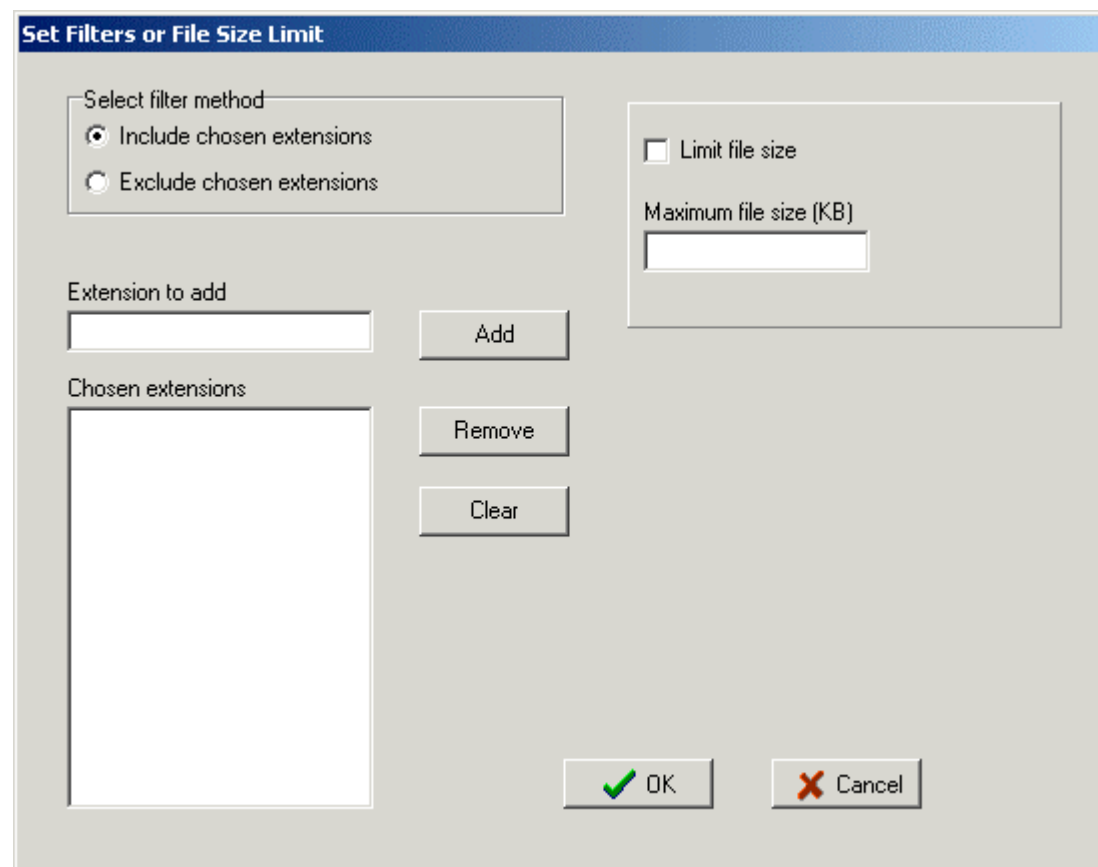
If the *DeleteFiles* property is set to true, the files will be deleted from the source folder. This option should not be used unless the files in the source folder are copies. The control does not verify that the files have been successfully uploaded. It assumes they have been uploaded if the remote script returns a 200 OK message.

4.4. Filters

When no filters are used, the entire contents of the source folder are listed for upload. This list can be reduced by specifying file extensions or a maximum file size. The user can configure filters by clicking the filter button . This opens the filter dialogue box.

Files can be filtered by extension, and a list of extensions can be added by entering them in the box and clicking Add. They can be selected and removed, or the list can be cleared. Extensions are added without the period character and wildcards are not permitted. The list of extensions can either represent files that are to be included, or files that are to be excluded, depending on the setting of the radio button.

Files can be filtered by setting a maximum size limit. The limit file size box must be checked and a value entered to specify the maximum size in KB.



The dialog box titled "Set Filters or File Size Limit" contains the following elements:

- Select filter method:** Two radio buttons, "Include chosen extensions" (selected) and "Exclude chosen extensions".
- Extension to add:** A text input field and an "Add" button.
- Chosen extensions:** A list box and buttons for "Remove" and "Clear".
- Limit file size:** A checkbox (unchecked) and a text input field labeled "Maximum file size (KB)".
- Buttons:** "OK" (with a green checkmark) and "Cancel" (with a red X) at the bottom right.

The filters can also be set programmatically and the filter dialogue box can be concealed from the user by setting *ShowFilterButton* to false. The properties relevant to filters are *FilterByInclusion*, *LimitFileSize* and *MaxFileSize*. The methods are *AddFilter* and *ClearFilters*.

4.5. Upload, Abort, Continue and Reset

The buttons at the bottom left of the control are enabled or disabled as appropriate. The Upload button becomes enabled when some files are listed in the files selected box. Clicking this button starts the

upload. For each file a post operation is created and sent to the remote URL complete with the form variables. This procedure repeats until all the files have been sent, an error is encountered, or the Abort button is pressed. If the upload is aborted by the user it can be continued by pressing Continue, but any file that has been partly uploaded will be started again. Pressing Reset refreshes the listing in the files selected box and allows the upload process to start again from the beginning.

Upload and *Reset* are also methods that can be called from code instead of clicking the buttons.

Two progress bars show the overall progress and the individual file progress, unless they are hidden by setting *ShowProgress* or *ShowFileProgress* to false.

4.6. Stored Settings - csXPostUpload.ini

The property settings that can be changed by the user are stored in an inifile, csXPostUpload.ini. On Windows 2000 and later this file is stored in the application data folder for the current user under the subfolder "Chestysoft\csXPostUpload". Storing these settings prevents the user from having to update path and filter details every time they use the control. Settings that are changed programmatically will overrule the property settings from the inifile.

4.7. Server Authentication

Support for server authentication has been added in version 1.1. 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 *Username* and *Password* properties. If these properties are not set, a dialogue box will be used to collect the log in details from the user.

5. Hiding the Interface and Selecting Files by Name

It is possible to control `csXPostUpload` from the calling application and reduce the appearance of the control to a progress bar or even hide it completely. Instead of uploading the files from a specified directory files to upload can be selected by name.

The control can be hidden completely by setting the *Visible* property to false. Setting *ShowProgressOnly* to true will reduce the control in size and display the progress bars and Abort button only. This Abort button can be removed by also setting *ShowProgressOnlyNoAbort* to true. If the control is run in this way the *ShowMessages* property can be set to false to hide the pop up messages that are shown when there is an error.

The upload process can be started without user intervention by calling the *Upload* method. After a user abort uploading can be resumed by calling *Upload* again.

Files can be added by specifying their name and physical path in the *AddFile* method. The *ManuallyAddFiles* property must be set to true for these files to be used instead of those listed in the list box in the control itself. As files are uploaded they are removed from the list in the order that they were added. The *FileCount* property shows the number of files in this list and it will be zero after calling *Upload* if there were no errors and no user Abort. The *UploadStatus* property indicates what happened to the last file in the batch and it will be 200 after a successful upload, because that is the HTTP status code.

These methods and properties allow `csXPostUpload` to be used as a general file upload component that can be used in an application written in a COM enabled language such as Visual Basic or Delphi.

The following code could be used in Visual Basic to upload a file while hiding the control:

```
Upload1.Visible = False
Upload1.RemoteURL = "http://somesite/filesave.asp"
Upload1.ShowMessages = False
Upload1.ManuallyAddFiles = True
Upload1.AddFile "C:\files\filetoupload.zip"
Upload1.Upload
```

6. Saving Uploads on a Server

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.

It is recommended that the server side script is tested first using a simple HTML upload form because any errors can usually be found easily and corrected. When the script is being tested it is important to use files that are the same size as those to be uploaded with `csXPostUpload` because some scripting languages or server settings limit the sizes of uploaded files.

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.

6.1. ASP

There is no built in method for saving an HTTP upload in ASP and a third party component is required. We sell a component called [csASPUpload](#) and so we will describe how that would be used to save a file using ASP. If using a different component refer to the documentation.

```
<%@ language=vbscript %>
<%
    Set Upload = Server.CreateObject("csASPUpload.Process")
    Upload.FileSave Upload.CurrentDir & Upload.FileName(), 0
%>
```

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

6.2. ASP.NET

The `HTMLInputFile.PostedFile` method of saving an uploaded file in ASP.NET cannot be used with `csXPostUpload`. We sell a .NET class called [csNetUpload](#) that can be used in the ASP.NET script to save uploaded files. The following code would save a file using `csNetUpload`.

```
<%@ Page language="vb" %>
<%@ Import Namespace = "csNetUpload" %>
<%
Dim Upload As New UploadClass
Upload.ReadUpload
Upload.SaveFile(0, Server.MapPath("./") & Upload.Filename())
%>
```

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

6.3. 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="filesent" destination=#filename#>
```

The *FormTagName* property of `csXPostUpload` must be set to "filesent" to work with this script.

6.4. 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['filesent']['tmp_name'],
$_FILES['filesent']['name'])
?>
```

The *FormTagName* property of `csXPostUpload` must be set to "filesent" to work with this script.

7. Deploying an Application

There is a description here of how to deploy csXPostUpload with a compiled application, and with a web browser application. Note that both the full and trial versions are licensed controls and are supplied with a licence file, csXPostUpload.lic or csXPostUploadTrial.lic. The reason for the trial version having a licence file is so that any difficulties that this restriction may cause can be identified at the trial stage.

7.1. Compiled Applications (e.g Visual Basic or Delphi)

In order to deploy an application that uses csXPostUpload you will need to distribute the OCX file, csXPostUpload.ocx, together with the files that make up your application. This file will need to be registered on the machine running your application and you may wish to use a proprietary installer to do this. When csXPostUpload was installed on your system our installer will have copied the OCX file to the directory "Program Files\Chestysoft\csXPostUpload\", assuming you accepted the defaults.

The number of copies of the OCX file that may be distributed is not limited by the licence. In order to use the control in a design environment the licence file, csXPostUpload.lic, is also required. The number of machines this may be installed on is governed by your licence agreement and it must not be copied to any more machines than permitted by the licence.

7.2. Web Browser Applications (e.g. Javascript)

In order for the control to run inside a web page the OCX file, csXPostUpload.ocx, must be present and registered on the client machine and the licence file, csXPostUpload.lic must be present. The OCX file can either be included in the <OBJECT> tag using the CODEBASE attribute and downloaded from the server, or it can be copied separately onto the client machine. If copied separately it must also be registered either by using a proprietary installer, by running regsvr32.exe at the command prompt or by using our free utility [DLLRegSvr.exe](#). If it is downloaded through the codebase attribute it can be compressed into a .CAB file to reduce the size of the download.

The licence file cannot be distributed royalty free but Microsoft have provided a method of sharing the licence file from the server. This is done by producing a licence package file (.lpk) using the Microsoft LPK File Generation. Refer to Section 1.1 for details of creating and using a licence package file.

A licence package file is supplied with the installer.

The csXPostUpload control is not digitally signed as safe. This means that the web browser security settings must be lowered to accept the installation of an unsigned ActiveX control. Once the control has been downloaded to the browser it will be stored on the local machine and the security restrictions can be raised again. It is not possible to sign this control because it accesses files in the local file system and for this reason it does not meet the definition of a "safe" control.

8. Other Products From Chestysoft

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

ActiveX Controls

[csXImage](#)

- An ActiveX control to display, edit and scan images.

[csXGraph](#)

- An ActiveX control to draw pie charts, bar charts and line graphs.

[csXThumbUpload](#)

- Upload multiple files by HTTP or FTP with previews and image edits.

[csXMultiUpload](#)

- Select and upload multiple files and post to a server using HTTP.

ASP Components

[csImageFile](#)

- Powerful server side image manipulation component.

[csDrawGraph](#)

- Draw pie charts, bar charts and line graphs in ASP.

[csASPGif](#)

- Create and edit animated GIFs.

[csIniFile](#)

- Read and Edit Windows style inifiles.

[csASPUpload](#)

- Process file uploads through a browser.

[csASPZipFile](#)

- Create zip files and control binary file downloads.

[csFileDownload](#)

- Control file downloads with an ASP script.

[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.](#)

9. Alphabetical List of Commands

Command	Page
About	6
AddFile	6
AddFilter	6
AddHiddenVariable	6
AddVariable	6
AuthenticationType	8
AutoSize	8
ClearFiles	6
ClearFilters	6
ClearHiddenVariables	6
ClearVariables	6
ContinueOnError	8
DeleteFiles	7
EnableLocalDestination	6
EnableSourceFolder	6
EnableURL	6
FileCount	7
FilterByInclusion	7
FormTagName	7
LimitFileSize	7
LocalDestination	6
ManuallyAddFiles	7
MaxFileSize	7
OnAbort	8
OnFileComplete	8
OnFileStart	8
OnUploadComplete	8
OnUploadStart	8
Password	8
RemoteURL	6
Reset	6
ReturnText	7
ShowFileProgress	7
ShowFilesSelected	7
ShowFilesSent	7
ShowFilterButton	6
ShowLocalDestination	6
ShowMessages	7
ShowProgress	7
ShowProgressOnly	7
ShowProgressOnlyNoAbort	7
ShowSourceFolder	6
ShowURL	6
ShowVariables	7
SourceFolder	6
Upload	6
UploadStatus	7
Username	8
Visible	8
WarnNoVariables	7