



Website: [www.chestysoft.com](http://www.chestysoft.com)

Email: [info@chestysoft.com](mailto:info@chestysoft.com)

## **csNetUpload - Version 1.0**

### **ASP.NET Class for Saving Uploaded Files**

This is a .NET class that is used to save files that have been uploaded to an ASP.NET application using an HTTP post. Typically uploaded files are sent by a web browser using the <input type=file> tag, which displays a "Browse" button for the user to select the file. csNetUpload can also be used when the files have been submitted using a client side control, such as the PostImage function of our csXImage control.

csNetUpload can save the uploaded file or files to disk, or export them as an array of bytes or as a MemoryStream object. The array of bytes can be used to save the file into a binary database field. The stream can be used with image files to pass the file into a Bitmap object or the Chestysoft csImageFile COM object.

A free, fully functional trial version of csNetUpload is available. This trial version has a built in expiry date that causes it to stop working after that time. This is the only functional limitation between the trial and full versions. This means that you can fully test if this component is suitable for your application before considering whether to license the full version.

### **Using these Instructions**

These instructions are divided into a number of sections with quick links to each section. 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:

- [Getting Started](#)
- [Saving or Exporting the Uploaded File](#)
- [Examples](#)
- [Server Settings and Configuration](#)
- [Alphabetical List of Commands](#)

# TABLE OF CONTENTS

<b>1. GETTING STARTED.....</b>	<b>3</b>
1.1. USING CSNETUPLOAD IN ASP.NET .....	3
1.2. THE TRIAL VERSION.....	3
<b>2. SAVING OR EXPORTING THE UPLOADED FILE.....</b>	<b>5</b>
2.1. METHODS FOR SAVING OR EXPORTING FILES .....	5
2.2. METHODS AND PROPERTIES FOR CHECKING UPLOADED FILES.....	5
2.3. OVERWRITE MODE .....	6
<b>3. METHODS FOR READING FILE PROPERTIES .....</b>	<b>7</b>
<b>4. EXAMPLES .....</b>	<b>8</b>
4.1. SAVING A SINGLE FILE .....	8
4.2. SAVING MULTIPLE FILES .....	9
4.3. SAVING THE FILE TO A DATABASE FIELD .....	10
4.4. COPYING THE FILE TO A BITMAP OBJECT .....	11
4.5. COPYING THE FILE INTO THE CSIMAGEFILE COM OBJECT .....	11
<b>5. SERVER SETTINGS AND CONFIGURATION .....</b>	<b>13</b>
5.1. LOCATION AND PERMISSIONS FOR THE DLL.....	13
5.2. PERMISSIONS FOR SAVING FILES .....	13
5.3. LIMITING THE SIZE OF FILE UPLOADS.....	13
<b>6. OTHER PRODUCTS FROM CHESTYSOFT .....</b>	<b>14</b>
<b>7. ALPHABETICAL LIST OF COMMANDS.....</b>	<b>15</b>

# 1. Getting Started

## 1.1. Using csNetUpload in ASP.NET

The DLL file Chestysoft.csNetUpload.dll (or Chestysoft.csNetUploadTrial.dll for the trial version) must be placed in the \bin folder for the web application. The ASP.NET machine account must be have Read and Execute permission on the DLL file.

The permissions on the \bin folder should not allow access for the anonymous internet user, IUSR\_machine\_name. This is to prevent users from downloading DLLs or components.

The ASP.NET script must import the namespace csNetUpload (csNetUploadTrial for the trial version). The class name is UploadClass and the *ReadUpload* method must be called to extract the files from the upload.

### Creating the component instance in VB.NET:

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

### Creating the component instance in C#:

```
<%@ Page language="c#" debug="true" %>
<%@ import Namespace = "csNetUpload" %>
<%
.
UploadClass Upload = new UploadClass();
Upload.ReadUpload();
.
%>
```

Both these examples show use with the full version of the class.

All the remaining examples are shown using VB.NET.

## 1.2. The Trial Version

The trial version of csNetUpload is supplied as a separate DLL called Chestysoft.csNetUploadTrial.dll. This trial version is fully functional but it has an expiry date, after which time it will stop working. An exception will be raised if an attempt is made to call the *ReadUpload* method after the expiry date.

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 in ASP.NET (VB):

```
Dim Upload As New UploadClass
Response.Write(Upload.Version)
```

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

The trial version has the namespace csNetUploadTrial, compared with csNetUpload in the full version. After upgrading to the full version the scripts using the class will need to have this namespace name changed.

## 2. Saving or Exporting the Uploaded File

Before files can be saved or exported, the *ReadUpload* method must be called.

<b>ReadUpload ( )</b> - This method extracts the file data from the HTTP post.
--------------------------------------------------------------------------------

There may be more than one file uploaded using HTTP, although frequently there is only one. In order to identify separate files in a multiple upload all the file methods have either an index parameter or a string parameter which we refer to as the *TagName* parameter. The index is zero based so the first file will have an index of zero, the second will have an index of 1, and so on. The *TagName* is the name attribute of the `<input type="file" name="TagName">` tag in the HTML form that posts the file.

In a multiple upload from a browser it is possible that the user will not select a file for each field. Only successfully uploaded files are given an index, so the index cannot be used to match a file with a particular form field.

To prevent runtime errors, the *FileCount* property can be used to find the number of files uploaded, which may be zero if none were selected by the user. The *FileUploaded* method can be used to find if a particular *TagName* has a file associated with it.

### 2.1. Methods for Saving or Exporting Files

<b>SaveFile (Index As Integer, Path As String)</b> - This saves the file indicated by <i>Index</i> to the location specified by <i>Path</i> . <i>Path</i> is the physical path on the server, complete with file name and extension.
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>SaveFile (TagName As String, Path As String)</b> - This saves the file indicated by <i>TagName</i> to the location specified by <i>Path</i> . <i>Path</i> is the physical path on the server, complete with file name and extension.
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>FileToArray (Index As Integer)</b> - This returns an array of bytes containing the file specified by <i>Index</i> .
------------------------------------------------------------------------------------------------------------------------

<b>FileToArray (TagName As String)</b> - This returns an array of bytes containing the file specified by <i>TagName</i> .
---------------------------------------------------------------------------------------------------------------------------

<b>FileToStream (Index As Integer)</b> - This returns a MemoryStream containing the file specified by <i>Index</i> .
----------------------------------------------------------------------------------------------------------------------

<b>FileToStream (TagName As String)</b> - This returns a MemoryStream containing the file specified by <i>TagName</i> .
-------------------------------------------------------------------------------------------------------------------------

### 2.2. Methods and Properties for Checking Uploaded Files

<b>FileCount</b> - Integer, read only property. This contains the number of files uploaded. It can be read before attempting to save a file or it can be used for looping if multiple files are uploaded.
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>FileUploaded (TagName As String)</b> - Boolean return value. This returns true if a file has been uploaded using the <i>TagName</i> specified.
---------------------------------------------------------------------------------------------------------------------------------------------------

<b>GetTagName (Index As Integer)</b> - String return value. Returns the <i>TagName</i> for the file specified by <i>Index</i> , or an empty string if there is no file for that index.
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>GetIndex (TagName As String)</b> - Integer return value. Returns the index for the file specified by <i>TagName</i> or -1 if there is no file.
---------------------------------------------------------------------------------------------------------------------------------------------------

## 2.3. Overwrite Mode

The *OverwriteMode* property can be used as an easy way to control whether existing files are overwritten by the *SaveFile* method.

<b>OverwriteMode</b>	-	Integer, 0, 1 or 2. This property determines what happens if the <i>SaveFile</i> method attempts to write to a file that already exists. The default value is 0.
<i>OverwriteMode</i> = 0	-	Any existing file will be overwritten.
<i>OverwriteMode</i> = 1	-	The new file will not be saved if an existing file has the same name.
<i>OverwriteMode</i> = 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 <i>OverwriteMode</i> of 2 is used, the file property <i>NewName</i> will be set to the name of the file that was actually saved.		

Example:

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

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

If an alternative character to the '~' is needed this can be assigned to the property *OverwriteChr*. This might be necessary on Windows 2003 or if URLScan is used because URLs containing this character may be blocked from downloading.

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

### 3. Methods for Reading File Properties

The following methods return information about uploaded files, such as the name, extension or file size. As with the methods for saving files there are two versions of each method, one using the file index to identify the file, the other using the *TagName*.

**FileName** (*Index* As Integer) - String. This returns the original file name of the uploaded file specified by *Index*. This includes the extension, but not the path.

**FileName** (*TagName* As String) - String. This returns the original file name of the uploaded file specified by *TagName*. This includes the extension, but not the path.

**Extension** (*Index* As Integer) - String. This returns the extension of the uploaded file specified by *Index*. It includes the period character.

**Extension** (*TagName* As String) - String. This returns the extension of the uploaded file specified by *TagName*. It includes the period character.

**LocalName** (*Index* As Integer) - String. This returns the full path of the file specified by *Index*, as used on the client computer before uploading. It is the value in the file field and can also be read using Request.Form.

**LocalName** (*TagName* As String) - String. This returns the full path of the file specified by *TagName*, as used on the client computer before uploading. It is the value in the file field and can also be read using Request.Form.

**ContentType** (*Index* As Integer) - String. This returns the content type that the sender indicated for the file specified by *Index*.

**ContentType** (*TagName* As String) - String. This returns the content type that the sender indicated for the file specified by *TagName*.

**FileSize** (*Index* As Integer) - Integer. This returns the file size in bytes for the file specified by *Index*.

**FileSize** (*TagName* As String) - Integer. This returns the file size in bytes for the file specified by *TagName*.

**NewName** (*Index* As Integer) - String, read only. If the file is renamed during saving to prevent overwriting, this property is set to the new file name. The saved file is specified by *Index*. See also the section on *OverwriteMode*.

**NewName** (*TagName* As String) - String, read only. If the file is renamed during saving to prevent overwriting, this property is set to the new file name. The saved file is specified by *TagName*. See also the section on *OverwriteMode*.

## 4. Examples

### 4.1. Saving a Single File

#### Example HTML form to upload a single file:

```
<form method="post" action="savefile.aspx" enctype="multipart/form-  
data">  
<input type="file" name="file1">  
<input type="submit" value="Send File">  
</form>
```

This form will allow the user to select a single file using the "Browse" button. It will post the file to a script called "savefile.aspx". It is important to set the enctype attribute to "multipart/form-data" when the form is uploading a file.

#### Saving a file - no error checking:

```
<%@ Page language="vb" debug="true" %>  
<%@ Import Namespace = "csNetUpload" %>  
<%  
Dim Upload As New UploadClass  
Upload.ReadUpload  
Upload.SaveFile(0, Server.MapPath("./files/") & Upload.Filename(0))  
Response.Write("File saved: " & Upload.FileName(0))  
%>
```

The VB.NET code shown above will save a single uploaded file into a sub directory called "files", using the original file name. It uses the index to specify the file to save, and with a single file this index is 0. If the *TagName* was used instead, and if the form shown earlier was used, the SaveFile command would become:

```
Upload.SaveFile("file1", Server.MapPath("./files/") &_  
Upload.Filename("file1"))
```

This example does not check for errors and will raise an "Index out of range" exception if the user does not submit a file. When the user is uploading the file through a form they will expect the errors to be handled gracefully, as in the next example. There are occasions when it is better to leave the error messages, for example when the Chestysoft csXImage control is used to upload an image it uses the HTTP return code to check for a successful upload and allowing the script to raise an exception will pass a 500 return code for an internal server error.

#### Saving a file - with error checking:

This is a modification on the previous example to check that a file has been uploaded before attempting to save it.

```
<%@ Page language="vb" debug="true" %>  
<%@ Import Namespace = "csNetUpload" %>  
<%  
Dim Upload As New UploadClass  
Upload.ReadUpload  
If Upload.FileCount > 0 Then  
    Upload.SaveFile(0, Server.MapPath("./files/") & Upload.Filename(0))  
    Response.Write("File saved: " & Upload.FileName(0))  
Else
```

```

    Response.Write("No file uploaded.")
End If
%>

```

The *FileCount* property is used to check if a file was uploaded and *SaveFile* is only called if there is a file to save.

If the *TagName* is being used to specify the file to save, the If statement would become:

```

If Upload.FileUploaded("file1") Then

```

## 4.2. Saving Multiple Files

### Example HTML form to upload 3 files:

```

<form method="post" action="savefile.aspx" enctype="multipart/form-
data">
<input type="file" name="file1">
<input type="file" name="file2">
<input type="file" name="file3">
<input type="submit" value="Send File">
</form>

```

This form will allow the user to select up to three files using three separate "Browse" buttons. It will post the files to a script called "savefile.aspx". Note that the web browser does not allow multiple file selection and this is why a separate file tag is required for each file.

Saving the files in a loop:

```

<%@ Page language="vb" debug="true" %>
<%@ Import NameSpace = "csNetUpload" %>
<%
Dim Upload As New UploadClass
Dim I As Integer
Upload.ReadUpload
If Upload.FileCount > 0 Then
    For I = 0 to Upload.FileCount - 1
        Upload.SaveFile(I, Server.MapPath("./files/") & _
Upload.Filename(I))
        Response.Write("File saved: " & Upload.FileName(I) & "<br>")
    Next
Else
    Response.Write("No file uploaded.")
End If
%>

```

This example checks the *FileCount* property to ensure that there is at least one file before running a For..Next loop to save each file. In each case it uses the original file name when saving. If the user misses one of the files, there will be no file with an index of 2. For this reason the index method of accessing the files is not suitable if the uploaded files need to be matched with the form fields. The next example uses the *TagName* to save each file.

### Saving the files using the TagName:

```

<%@ Page language="vb" debug="true" %>

```

```

<%@ Import Namespace = "csNetUpload" %>
<%
Dim Upload As New UploadClass
Dim I As Integer
Upload.ReadUpload
If Upload.FileCount > 0 Then
    If Upload.FileUploaded("file1") Then
        Upload.SaveFile(("file1"), Server.MapPath("./files/") & _
            "file1" & Upload.Extension("file1"))
        Response.Write("file1 saved." & "<br>")
    End If
    If Upload.FileUploaded("file2") Then
        Upload.SaveFile(("file2"), Server.MapPath("./files/") & _
            "file2" & Upload.Extension("file2"))
        Response.Write("file2 saved." & "<br>")
    End If
    If Upload.FileUploaded("file3") Then
        Upload.SaveFile(("file3"), Server.MapPath("./files/") & _
            "file3" & Upload.Extension("file3"))
        Response.Write("file3 saved." & "<br>")
    End If
Else
    Response.Write("No file uploaded.")
End If
%>

```

In this example the *FileCount* property is still used to check if the user has not submitted any files, so that a specific message can be shown. Each individual file is then checked and if it is present it is saved using a new name, but with the original extension.

### 4.3. Saving the File to a Database Field

In most cases, when a file is uploaded to the server it is saved on disk, as described above. Sometimes there is a requirement to write the file into a binary database field. The following example shows how a file can be written to a Microsoft Access database using a field type of OLE Object.

```

<%@ Page language="vb" debug="true" %>
<%@ Import Namespace = "System.Data.OleDb" %>
<%@ Import Namespace = "System.Data" %>
<%@ Import Namespace = "System.DateTime" %>
<%@ Import Namespace = "csNetUpload" %>
<%
Dim Upload As New UploadClass
Upload.ReadUpload
If Upload.FileCount > 0 Then
    Dim ConnectionString As String = "PROVIDER=MICROSOFT.JET.OLEDB.4.0;DATA
SOURCE=" & Server.MapPath("sample.mdb")
    Dim AConnection As OleDbConnection = New OleDbConnection(ConnectionString)
    Dim SQLString As String = "SELECT * FROM Table1 WHERE 1=2"
    Dim DA As OleDbDataAdapter = New OleDbDataAdapter(SQLString,
ConnectionString)
    Dim CommandBuilder As OleDbCommandBuilder = New OleDbCommandBuilder(DA)
    Dim DS As DataSet = New DataSet("Table1")
    DA.MissingSchemaAction = MissingSchemaAction.AddWithKey
    DA.Fill(DS, "Table1")
    Dim Row As DataRow = DS.Tables("Table1").NewRow
    Row("FileName") = Upload.FileName(0)
    Row("FileData") = Upload.FileToArray(0)
    Row("ContentType") = Upload.ContentType(0)
    Row("Extension") = Upload.Extension(0)
    Row("UploadDate") = DateTime.Now

```

```

    DS.Tables("Table1").Rows.Add(Row)
    DA.Update(DS, "Table1")
    AConnection.Close
    'File saved
Else
    'No file uploaded
End If
%>

```

At lot of the code is creating the various classes that are used to open and write to the database. It must use a data adapter in order to handle complex data types, such as the OLE Object. The database fields "FileName", "ContentType" and "Extension" contain information about the file and "UploadDate" contains the current date. "FileData" is the binary data and this can be written directly using the *FileToArray* method from csNetUpload.

This example is available for download, together with scripts for viewing the data from the database and for deleting the entries - <http://www.chestysoft.com/netupload/dbdemo.asp>.

## 4.4. Copying the File to a Bitmap Object

The *FileToStream* method can be used to copy uploaded images into a System.Drawing.Bitmap class. The following code checks the extension of the uploaded image and if it is .bmp or .jpg it loads the image and writes out the width and height.

```

<%@ Page language="vb" debug="true" %>
<%@ Import Namespace = "csNetUpload" %>
<%@ Import Namespace = "System.Drawing" %>
<%
Dim Upload As New UploadClass
Upload.ReadUpload
If Upload.FileCount > 0 Then
    If (Upload.Extension(0) = ".bmp") or _
        (Upload.Extension(0) = ".jpg") Then
        Dim BMP As New Bitmap(Upload.FileToStream(0))
        Response.Write("Width: " & BMP.Width & "<br>")
        Response.Write("Height: " & BMP.Height & "<br>")
    End If
Else
    Response.Write("No file uploaded.")
End If
%>

```

Once the image has been loaded into a Bitmap object, further image processing can be done before saving the image. Other file types are supported by the Bitmap object but only jpeg and bmp have been used here for simplicity.

## 4.5. Copying the File into the csImageFile COM Object

Image files can be copied directly into the Chestysoft csImageFile COM object, using similar code to that shown in Section 4.4. The *FileToArray* method is used.

```

<%@ Page language="vb" debug="true" %>
<%@ Import Namespace = "csNetUpload" %>
<%@ Import Namespace = "System.Drawing" %>
<%
Dim Image = Server.CreateObject("csImageFile.Manage")

```

```

Dim Upload As New UploadClass
Upload.ReadUpload
If Upload.FileCount > 0 Then
    If (Upload.Extension(0) = ".bmp") or _
        (Upload.Extension(0) = ".jpg") Then
        Image.ReadVariant(Upload.FileToArray(0))
        Response.Write("Width: " & Image.Width & "<br>")
        Response.Write("Height: " & Image.Height & "<br>")
    End If
Else
    Response.Write("No file uploaded.")
End If
%>

```

The *Extension* function can be used to check if the file is in a valid format, and if so, it can be loaded using the *csImageFile ReadVariant* function. As with the Bitmap example in the previous section, the width and height are displayed, but further image processing can be carried out before saving the image. Other file types are supported by *csImageFile* but only jpeg and bmp have been used here for simplicity.

## 5. Server Settings and Configuration

This section summarises the configuration settings that must be made on the server to run csNetUpload.

### 5.1. Location and Permissions for the DLL

The DLL file, Chestysoft.csNetUpload.dll (Chestysoft.csNetUploadTrial.dll for the trial version) must be located in the binary folder for the web application. By default, this folder is called "\bin". The permission settings on the DLL must allow the ASP.NET machine account Read and Execute permission. The Internet Guest User account (IUSR\_machine\_name) should not be allowed to read the DLL, to protect it from unauthorised downloads.

### 5.2. Permissions for Saving Files

The ASP.NET machine account must have Write permission in the folder where the uploaded files are to be saved. In addition, some consideration should be made to security. If the uploaded files are saved in a location that can be viewed by web users, it may be advisable to disable the use of scripts and executables in this folder. Otherwise a user would be able to upload a script or other executable file and then run it.

### 5.3. Limiting the Size of File Uploads

There is a size limit on file uploads, which is determined by the web application configuration. The size limit is defined by the maxRequestLength attribute of the httpRuntime element in either the Machine.config or Web.config files. The default value is 4 MB and usually this value is left in the Machine.config file and a Web.config file is placed in the application root folder, or the script folder, with a new value to override the default.

Here is an example of the code to be placed inside a Web.config file to set the upload size limit to 10 MB.

```
<configuration>
  <system.web>
    <httpRuntime maxRequestLength="10240" />
  </system.web>
</configuration>
```

The value is measured in kilobytes.

When a file is uploaded which is over the size limit, it produces a "Cannot find server or DNS error" message. This is not user friendly and it might be preferable to set the maxRequestLength to a value that is larger than any expected file size and if the file size needs to be limited to a smaller value, read the Request.TotalBytes property to find the uploaded size before calling csNetUpload.

## 6. 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.
- [csXPostUpload](#) - Uploads batches of files from a client to a server using an HTTP post.
- [csXMultiUpload](#) - Select and upload multiple files and post to a server using HTTP.

### ASP Components

- [csImageFile](#) - Resize, edit and create images in ASP.
- [csDrawGraph](#) - Component to draw pie charts, bar charts and line graphs.
- [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

- [csNetDownload](#) - ASP.NET class to control file downloads.
- [csASPNetGraph](#) - ASP.NET component to draw pie charts, bar charts and line graphs.

### Web Hosting

We can offer ASP/ASP.NET enabled web hosting with our components installed. [Click for details.](#)

## 7. Alphabetical List of Commands

Command	Page no.
ContentType	7
Extension	7
FileCount	5
FileName	7
FileSize	7
FileToArray	5
FileToStream	5
FileUploaded	5
GetIndex	5
GetTagName	5
LocalName	7
NewName	7
OverwriteChr	6
OverwriteMode	6
ReadUpload	5
SaveFile	5
Version	3