w3 Upload 2.0

User Manual

Contents

Copyright

Dimac Development
Duplo AB
Prästgatan 12

252 24 Helsingborg
Sweden

Phone: +46 42 35 94 00
Fax: +46 42 15 80 50
Homepage: http://www.dimac.net
E-mail: info@dimac.net
Support: support@dimac.net

If any problems occur using this application, please visit our web site http://www.dimac.net.

Introduction

w3 Upload makes it possible for you to upload multiple files using simple ASP and HTML-coding technologies, and to control the maximum file size or to limit the content type of the uploaded files. The placement of uploaded files can be directed and named by the user who. w3 Upload allows you to mix standard forms with the upload form, thus making it possible for you to seamlessly integrate w3 Upload into your forms. It also provides the possibility to create and extract content from ZIP files.

Getting Started

Uploading to a server

w3 Upload works exactly as the usual HTTP form object and the file is accessed from the server for storage or manipulation as any other field (text for example). To utilize this architecture you need two pieces of code: The form where the client specifies and submits the file and receiving code which saves the file on the server or generates any other response.

The following is an example of a very simple form that could be used:

```
<%@ LANGUAGE="JAVASCRIPT" %>
<HTML>
<HEAD>
    <TITLE> Upload Form </TITLE>
</HEAD>
 <BODY>
 <H1>Load Form</H1>

 <FORM ACTION="loadit.asp" METHOD="POST" ENCTYPE="multipart/form-data">
          Name:<BR>
          <INPUT NAME="fileName"><BR>
          File:<BR>
          <INPUT NAME="theFile" TYPE=File><BR>
          <INPUT TYPE="Submit" VALUE="Send!"><BR>
 </FORM>

 </BODY>
</HTML>
```

Things to note:

- The ‘ENCTYPE’ parameter of the ‘FORM’ tag: Has to be set to `"multipart/form-data"` as specified
- There are two fields submitted: (1) A field for specifying the name the user wishes (of type ‘text’, which is default and not explicitly declared in the ‘INPUT’ tag). (2) A field for the file where the user specifies the search path to the file either by keyboard entry or using the standard windows file browse dialog.
- As stated previously, the client must be using a browser that supports the ‘INPUT’ tag of ‘FILE’ type.

IIS 6.0: upload and download issues

If you are having issues with uploads and downloads with file sizes greater than 200K and are running IIS 6.0, try these solutions first: open up the file "metabase.xml" with Notepad. It should be located in C:\windows\system32\inetserv. Find the variable named "AspMaxRequestEntityAllowed". By default it is set at 204800 bytes (200 Kb), change it to the desired size. For downloading Find the variable named "AspBufferingLimit" and change it to the desired value. By default this is set to 4 MB. By changing this variable you are increasing the max size of ASP response buffer, thus letting you download files greater than 4 MB. By increasing the number of these variables, you can control the size of uploads and downloads respectively.

A NOTE: Before making changes to this file, you must first go to IIS and right click the server and select properties. Check the box that says "allow changes to MetaBase configuration while IIS is running". If you don't, you will find that the metabase.xml file is locked. For good house cleaning you should go back and uncheck this after making your changes.

Object Model

W3.Upload

GetData

Signature:

```
      GetData( ) : String
```

Description:
Returns ALL mimedata as a string.

SendFile

Signature:

```
      SendFile(bstrFileName, [bSetContentType]) :
```

Description:
Sends a local file on the server to the client.

The method takes two arguments - path of the file to be sent and a content type. Instead of explicitly setting the content type you can set the second parameter to 'true' and leave it up to upLoad, which is probably the best way to do it.
Please see the 'Getting Started' chapter for usage information.

Example:

```
upLoad.SendFile( 'D:\\upload\\test.jpg', true );
// will send the specified file on the server to the client (browser),
// where it will most likely be displayed
```

Form

Signature:

```
      Form( ) : w3.FieldCollection
```

Description:
Replacement Form object. Use this as the normal form object. See also the 'Getting Started' chapter.

There are several uses as this enables access to the field collection of the submitted form. Perhaps is the creation of references to indivdual field objects of the collection the most common usage. You can either reference the field by it's name, by using a numeric index, or by enumeration (for each ..)
See the reference of your chosen development envirnoment on handling of collections for further information.

Example:

```
// creating a variable reference to an individual field object in the
field collection
// of the submitted form:

var aName = upLoad.Form( 'chosenFileName' );
var uploadedFile = upLoad.Form( 'theFile' );

// you can type-cast the variables if you wish, for readability reasons,
although it is usually an
// unecessary encapsulation of the object
// thus, alternatively, you could write:

var aName = new String ( upLoad.Form( 'chosenFileName' ) );
```

ProgressParts

Signature:

```
        ProgressParts : Long
```

Description:
Specifies how many intervals the progressbar should use. See **W3.Progressbar** for more information.

SendAdodbField

Signature:

```
        SendADODBField(AdoDBField as Object) :
```

Description
Deprecated. Please use FormField.BinaryData in conjunction with a stream object.
See FormField.BinaryData for more information.

W3.FieldCollection

The FieldCollection refers to the form items in the posted form.

Count

Signature:

```
        Count() : Integer
```

Description:
Returns the number of fields in the submitted form.

Useful if you wish to have a dynamic number of fields submitted.

Example:

```
var noOfFields = new Number ( upLoad.Form.Count );
// noOfFields contains the number of fields in the submitted form
```

Items

Signature:

```
      Items(index) : w3.FormField
```

Description:
Returns the field object.

This is the second way to create a variable reference to a field object of the submitted field collection. See also the 'Form' member of the w3.upload object.

Example:

```
var firstField = upLoad.Form.Items( 0 );
// firstField now contains the first field object of the submitted form

For each fItem in upload.Form  // loop the form using for each.
      Response.write fItem & " : " & upload.Form(fItem)
next
```

W3.FormField

IsEmpty

Signature:

```
      IsEmpty( ) : Boolean
```

Description:

Returns true if the field is empty.

Note: Not yet implemented.

SaveAsUniqueFile

Signature:

```
      SaveAsUniqueFile(bstrFileName) : String
```

Description:

Saves the file as a Unique file and returns the new name. Adds a numbering at the end of the given filename.

Same as 'SaveToFile' except for this numbering feature.

Example:

```
// In the receiving code for a HTTP form the following saves a
// file submitted in the field named 'theFile'
// a reference to a field object is created

file = upLoad.Form( 'theFile' );
Var theName =
file.SaveAsUniqueFile("D:\\uploadSite\\uploadedfile.txt" );

// the field (the file) is saved on the server in the catalog:
// "D:\uploadsite\ "
// with the name "uploadedfile.txt" or "uploadedfile(1).txt" if
// it already exists,
// or "uploadedfile(2).txt" and so on
// it returns the name into variable 'theName'
```

SaveToFile

Signature:

```
SaveToFile(bstrFileName) :
```

Description:

Saves the field to disk. The most obvious usage is for saving an uploaded file on the server and thus applying the method on a field of 'FILE' type.

Example:

```
// In the receiving code for a HTTP form the following saves a file
// submitted in the
// field named 'theFile'

// a reference to a field object is created

file = upLoad.Form( 'theFile' );
file.SaveToFile( "D:\\uploadSite\\uploadedFiles\\uploadedfile.txt" );

// the field (the file) is saved on the server in the catalog:
// "D:\uploadsite\uploadedFiles\"
// with the name "uploadedfile.txt"
```

ContentType

Signature:

```
ContentType( ) : String
```

Description:

Returns the contentType of the uploaded file. Only works on file fields.

Example:

```
var fileField = upLoad.Form( "theFile" );
var type = new String ( fileField.ContentType );
// type now contains the content type of the file
// for example: 'image/pjpeg'
```

FileName

Signature:

```
FileName( ) : String
```

Description:

The name of the originally uploaded file

Example:

```
var fileField = upLoad.Form( "theFile" );
var fileName = new String ( fileField.FileName );
// fileName contains the name of original file, including full path,
// as found on
// the computer. For example: 'D:\pictures\mypicture.jpg'
```

IsFile

Signature:

```
IsFile( ) : Boolean
```

Description:

Returns true if the field is a File

Example:

```
var fileField = upLoad.Form( "theFile" );
var isFile = new Boolean ( fileField.IsFile );
// isFile is true when the field is of file type
```

Item

Signature:

```
Item( ) : String
```

Description:

Value of this field. Returns the string if it is a textfield and so on.

Example:

```
var fileName = upLoad.Form( "theFileName" );
var value = new String ( fileField.FileName );
// value now contains the name (the input field 'theFileName' is of text
type)
```

Name

Signature:

```
Name( ) : String
```

Description:

Returns the name of the inputfield as specified in the HTML of the submitting formpage.

Example:

```
var fileField = upLoad.Form.Items( 1 );
var fieldName = new String ( fileField.Name );
//  fieldName  now  contains  the  name  of  the  field  -  in  this  case:
'theFile'
```

Size

Signature:

```
      Size( ) : Integer
```

Description:

Returns the size of the field/file

Example:

```
var fileField = upLoad.Form( "theFile" );
var fileSize = new Number ( fileField.Size );
// fileSize now contains the size of the file
```

Count

Signature:

```
      Count( ) : Integer
```

Description:

Returns the number of fields which share the same name in the collection.

Example:

```
var fileField = upLoad.Form( "theFile" );
var fileCount = new Number ( fileField.Count );
// fileCount is the number of fields with the name “theFile”
```

Field

Signature:

```
      Field(index) : w3.FormField
```

Description:

Returns the w3.FormField in the collection identified by the numeric index.
When multiple fields with the same name exists, the collection accessed by this property contains all the relevant w3.FormField objects.
The first (index 0) item actually represents the current object used.

Example:

```
var fileFields = upLoad.Form( "theFile" );
var i;
for( i = 0 ; i < fileFields.Count ; ++I )
{
  f = fileFields.Field( I );
  f.SaveAsUniqueFile(“upload.txt”);
}
```

BinaryData

Signature:

```
      BinaryData :  Byte[]
```

Description:

Returns a file as a byte array, useful when adding the file to a database.

Example:

```
rs = Server.CreateObject("ADODB.Recordset");
rs.Open("SELECT * FROM blobtest WHERE id=42", conn, adOpenDynamic,
adLockOptimistic);
rs("blob").AppendChunk(upload.Form("theFile").BinaryData);
rs.Update();
```

W3.Archive

Create

Signature:

```
      Create(fileName : String)
```

Description:

Creates a new archive with the specified file name.

Example:

```
archive.Create("c:\files\zipfile.zip");
```

Open

Signature:

```
      Open(FileName : String, [password : String]) : Bool
```

Description:

Returns a file as a byte array, useful when adding the file to a database.

Example:

```
arch.Open("c:\files\zipfile.zip", "s3cr3t");
```

Extract

Signature:

```
      Extract(zipFile  :  String,  destinationDir  :  String,  [fileMask  :
String], [pathType : Long])
```

Description:

Extract the files from the archive.

Example:

```
arch.Extract("c:\files\file.zip", "c:\newfiles", "*.*",
arch.RELATIVE_PATH);
```

ExtractFile

Signature:

```
      ExtractFile(fileName : String, destination : String)
```

Description:

Extracts a file from the archive. Note: the archive must be open.

Example:
```
arch.Open("c:\files\zipfile.zip");
arch.ExtractFile("w00t.jpg", "c:\files");
```

AddFile

Signature:
```
AddFile(fileName : String)
```
Description:

Adds a file to the archive. Note: the archive must be open.

Example:
```
arch.Open("c:\files\zipfile.zip");
arch.AddFile("c:\files\ls.txt");
```

AddFiles

Signature:
```
AddFiles(path : String, recurse : Bool)
```
Description:

Adds a directory to the archive. Note: the archive must be open.

Example:
```
arch.Open("c:\files\zipfile.zip");
arch.AddFiles("c:\files\*.*", true);
```

FindFile

Signature:
```
FindFile(fileName : String) :  Bool
```
Description:

Returns true if fileName could be found in the archive. Note: the archive must be open.

Example:
```
arch.Open("c:\files\zipfile.zip");
arch.FindFile("w00t.jpg");
```

Count

Signature:
```
Count :  Long
```
Description:

Returns the number of items in the zipfile.

Example:
```
arch.Open("c:\files\zipfile.zip");
Response.Write(arch.Count);
```

W3.ArchiveItem

Name

Signature:
```
Name :  String
```
Description:

Returns the name of the item.

Example:

```
arch.Open("c:\files\zipfile.zip");
for (var i=0; i < arch.Count; i++){
      Response.Write(arch.Item(i).Name + "<br>");
      Response.Write(arch.Item(i).Size);
}
```

Name

Signature:

```
      Name :  String
```

Description:

Returns the name of the item.

Example:

```
arch.Open("c:\files\zipfile.zip");
for (var i=0; i < arch.Count; i++){
      Response.Write(arch.Item(i).Name + "<br>");
      Response.Write(arch.Item(i).Size);
}
```

IsFolder

Signature:

```
      IsFolder :  Bool
```

Description:

Returns whether the item is a folder.

Example:

```
arch.Open("c:\files\zipfile.zip");
for (var i=0; i < arch.Count; i++){
      Response.Write(arch.Item(i).isFolder + "<br>");
}
```

Size

Signature:

```
      Size :  Long
```

Description:

Returns the uncompressed size of the item.

Example:

```
arch.Open("c:\files\zipfile.zip");
for (var i=0; i < arch.Count; i++){
      Response.Write(item.Size + "<br>");
}
```

ModifiedDate

Signature:

```
      ModifiedDate :  Date
```

Description:

Returns the modified date of the item.

Example:

```
arch.Open("c:\files\zipfile.zip");
for (var i=0; i < arch.Count; i++){
      Response.Write(item.ModifiedDate + "<br>");
}
```

W3.Progress

ProgressID

Signature:

```
ProgressID :  String
```

Description:

Returns a unique identifier used to identify an upload process.

Example:

```
Response.Write(progress.ProgressID);
```

GUID

Signature:

```
GUID :  String
```

Description:

Returns a GUID.

Example:

```
Response.Write(progress.GUID);
```