

Internet Extra 10

Receiving File Uploads with ASP.NET

If you need more flexibility or greater security than the Microsoft Office FrontPage 2003 Save Results component provides for uploading files, you might decide it's worthwhile to program your own Web page. This is the approach you must take if, for example, you want the same transaction to upload a file and update a database.

Processing uploaded files was fairly difficult using classic ASP and generally required installing third-party ActiveX components on the Web server. Microsoft ASP.NET provides built-in support for processing uploaded files, but there are several details you must get right:

- **The <form> tag** As with all <input> tags, you must code the <input type="file"> tag within an HTML form (that is, between a set of <form> and </form> tags). In addition, the <form> tag must specify the attributes *method="POST"* and *enctype="multipart/form-data"*. In FrontPage, these properties appear in the Options For Custom Form Handler dialog box.
- **The <input type="file"> tag** All recent Web browsers (specifically, those that support HTML 3.2 and later) support an <input type="file"> tag designed for uploading files to the Web server. In a Web page, this tag looks like an ordinary text box with an ordinary button to its right. To create such a tag in FrontPage, choose Form from the Insert menu, and then File Upload.
- **The visitor entry** To upload a file, the visitor must either hand-type the file's path and name or click the Browse button to locate the file by navigation. For security reasons, the Web server can never specify the file to upload.
- **The Web server** Just because a visitor uploads a file doesn't mean that the server saves it to the server's file system. The server saves the uploaded file only in memory, and only until it finishes processing the visitor's request.

If the request triggers a server-side program (an ASP.NET page, for example), that program can inspect the file in memory and write it into the server's file system.

This provides you with control over what data enters the server and where it goes.

When you use ASP.NET code to receive and process the file, there are two additional requirements. First, both the <form> tag and the <input type="file"> tag must both contain *runat="server"* attributes. Second, the <input type="file"> tag must contain an *id* attribute. Here, then, is the minimal HTML for a Web form that uploads a file:

```
<form method="POST" enctype="multipart/form-data" runat="server">
  <input type="file" id="fupUpload" runat="server" />
  <input type="submit" value="Upload" runat="server" />
</form>
```

Microsoft Office FrontPage 2003 Inside Out

When an `<input type="file" runat="server" />` tag runs on the server, it creates an *HtmlInputFile* object. This type of object supports all the normal properties and methods of any other Web control, as well as the properties in Table IE10-1.

Table IE10-1. Unique *HtmlInputFile* Properties

Property	Type	Mode	Description
PostedFile	HttpPostedFile	Read	The content of the uploaded file
Accept	String	Read/Write	A comma-delimited list of MIME types that constrains the file types the visitor can select
MaxLength	Integer	Read/Write	The maximum acceptable length (including the path) for the name of the uploaded file
Size	Integer	Read/Write	The width of the text box where the visitor enters the name (including the path) of the file to upload

You can specify the *Accept*, *MaxLength*, and *Size* properties either as HTML attributes or with code. Browser support for the *Accept* property varies widely; Microsoft Internet Explorer seems to ignore it completely. There's seldom a need to code the *MaxLength* property, but the *Size* property is useful for controlling how many characters the text box portion of the control displays. Visitors appreciate having the entire path and file name visible.

The *PostedFile* property is accessible only from code and useful only after a postback. If the form submission included a file, the *PostedFile* property will point to an *HttpPostedFile* object. If the form submission didn't include a file, the *PostedFile* property points to the reserved value *Nothing*. Consider, for example, the following file upload tag (which appeared in the previous code snippet):

```
<input type="file" id="fupUpload" runat="server" />
```

To test whether this object contains an uploaded file, you would code as follows:

```
If fupUpload.PostedFile Is Nothing Then
'   There's no file.
Else
'   There is a file.
End If
```

If an uploaded file is present, the properties in Table IE10-2 are available. The *fupUpload.PostedFile.ContentLength* property, for example, contains the size of the file in bytes (which, by the way, can be zero). The *fupUpload.PostedFile.FileName* property contains the fully qualified file name from the visitor's computer.

Receiving File Uploads with ASP.NET

Table IE10-2. *HttpPostedFile* Properties

Property	Description
ContentLength	Gets the size in bytes of an uploaded file
ContentType	Gets the MIME content type of a file sent by a visitor
FileName	Gets the fully qualified name of the file on the visitor's computer (for example, "C:\MyFiles\Test.txt")
InputStream	Gets a <i>Stream</i> object that points to an uploaded file to prepare for reading the contents of the file

The *InputStream* property temporarily holds the actual file contents. A *Stream* is an abstract object that behaves in most respects like an ordinary flat file. If the *InputStream* property contains a text file, for example, a *StreamReader* can open the file and read it. Here's some code that opens the uploaded file for input and displays it in a label control named *lblListing*:

```
Dim srdUpload As StreamReader
Dim strUpload As String

lblListing.Text = ""
srdUpload = New StreamReader(fupUpload.PostedFile.InputStream)
Do
    strUpload = srdUpload.ReadLine()
    If strUpload Is Nothing Then
        Exit Do
    End If
    lblListing.Text &= strUpload & "<br>" & vbCrLf
Loop
```

If the uploaded file is supposed to be a picture, you can try loading it into a *System.Drawing.Image* object, as shown in the following code. If this throws an exception, the picture file is invalid. If loading the picture is successful, you can check its height, width, format, and other properties by inspecting the properties of the *imgUpload* object.

```
Dim imgUpload As System.Drawing.Image

imgUpload = System.Drawing.Image.FromStream( _
    fupUpload.PostedFile.InputStream)
```

Although eminently useful, the *PostedFile.InputStream* object is fleeting. It disappears from memory the moment your ASP.NET page terminates. To save the file to a disk, you must call the *PostedFile* object's *SaveAs* method and specify a fully qualified physical path and file name, as shown here:

```
fupUpload.PostedFile.SaveAs( _
    "c:\InbetPub\wwwroot\webdbpgm\ch13\pix\newpic.gif")
```

Microsoft Office FrontPage 2003 Inside Out

Don't try to save the file in the *PostedFile.FileName* location. This specifies a path that existed on the visitor's computer, and such a path is almost certainly incorrect on the Web server. Instead, append the base file name to a server path you know is correct. The following statement, for example, saves a file with the base file name and extension the visitor specified, but in your application's uploads folder:

```
fupUpload.PostedFile.SaveAs( _
    Server.MapPath(Request.ApplicationPath & _
    "/uploads/" & _
    Path.GetFileName(fupUpload.PostedFile.FileName)))
```

If you installed the Insider Extra files from the companion CD, you'll find a sample page in the fileupload folder that processes uploaded files. The name of the file is fileupload.aspx, and a screen shot appears in Figure IE10-1.

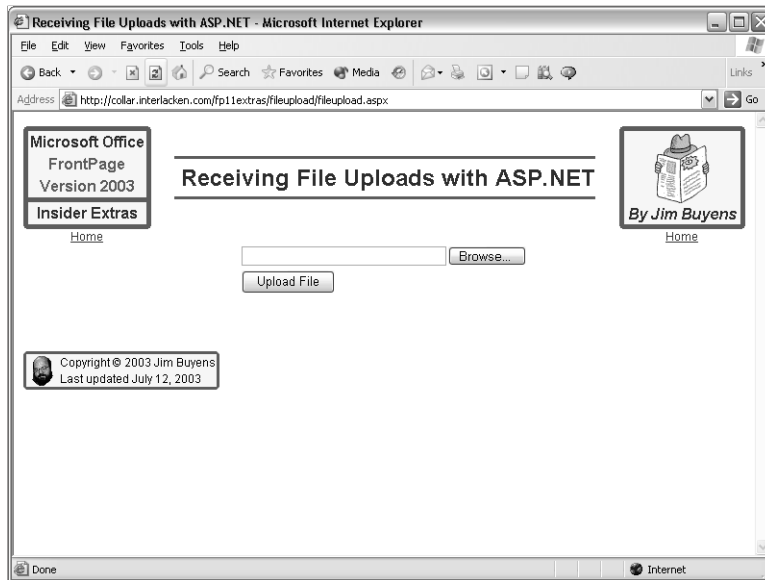


Figure IE10-1. This ASP.NET page uses custom programming to save any file the visitor uploads.