



LEI Http File Transfer Component User Guide

October 3, 2005

James H. Byrd
LHFTC_UserDoc.doc

Table of Contents

Revision History.....	2
Abstract	3
Installation.....	3
HttpDownload Class Guide	3
Setting Up a Download Folder	4
ASP Example	4
Login.asp.....	5
FileList.asp.....	7
DownloadFile.asp	9
Visual Basic Example.....	11
Configuring Your VB Project	11
Using the HttpDownload Class	12
HttpDownload Class Reference.....	13
Properties	13
Methods.....	14

Revision History

Date	Author	Description
10/2/2005	James H. Byrd	Document version 1 completed.

Abstract

This document describes the Logical Expressions HTTP File Transfer Component (FTC). The FTC is a COM class library with classes that let you transfer data between the server and the client browser using the HTTP protocol.

The FTC provides the following benefits:

- Allows you to easily create web applications that can transfer a server file to the client browser using the standard Windows Save As dialog with a minimum of coding.
- Supports the web applications you build with pure ASP and with Visual Basic.
- Allows you to prevent the theft of your content by letting only authenticated users have access to your files.
- Capable of transferring any kind of file, including the popular ZIP and PDF formats.

The Logical Expressions HTTP File Transfer Component is a product of Logical Expressions, Inc. By installing this product, you agree to abide by the Logical Expressions End User License Agreement, which is included in the installation package. You must review and agree to the license before you install and use this software.

Installation

Before you can use the transfer component library, you must install it on the Web server that hosts your site. This operation typically requires someone with administrative access to the server, so you may need the cooperation of your Web host. If your hosting company is concerned about installing third-party components on their server, please refer them to this document for information about what the component does.

In most cases, a full installation is not required. You can often just copy the component DLL onto the server and register it. Most modern systems already have the Visual Basic runtime loaded on them, so the component will run just fine. You can try that approach first, and if you have problems, do the full install.

The installation package includes a short and simple setup program that installs the HttpFileTransfer DLL. Once the DLL is registered on your server, you can make use of the classes it contains. Note that the setup program may require a system reboot to update the installation components it uses.

HttpDownload Class Guide

For most file types, the easiest way to offer a downloadable file is to put the files in a folder under your web site's root folder and simply link to them from one of your web pages. However, you don't always want to let everyone who visits your site have access to these files.

If you need to restrict access to downloadable content on your site, the files can't be in a folder under the Web site root. Any person who knows or discovers the URL of the file can retrieve it. This situation causes real problems for shopping sites that sell digital products and subscription content sites, to name just a couple of examples.

This is where HttpDownload comes in. This class lets you transfer files to a client browser from any folder on the web server or the network. Before granting access to a file, your application verifies that the user is authorized to retrieve it. Once you determine that the user is authorized, you invoke the HttpDownload class' Transfer method to transmit the file.

Setting Up a Download Folder

Note that you may need to negotiate with your Web site host to put your download files into a folder off the Web site. In many cases, the FTP connection your host sets up for you only gives you access to the Web site root folder and below. A better configuration is for them to give you FTP access to a higher level customer folder that has the web site's root folder *under* it. Then you can create a structure that looks something like this:

```
MyCustomerFolder
...MyDownloadFiles
...MyWebSite
.....images
```

You put all of your Web site files into MyWebSite, and you put all of your download files into MyDownloadFiles, as the structure indicates.

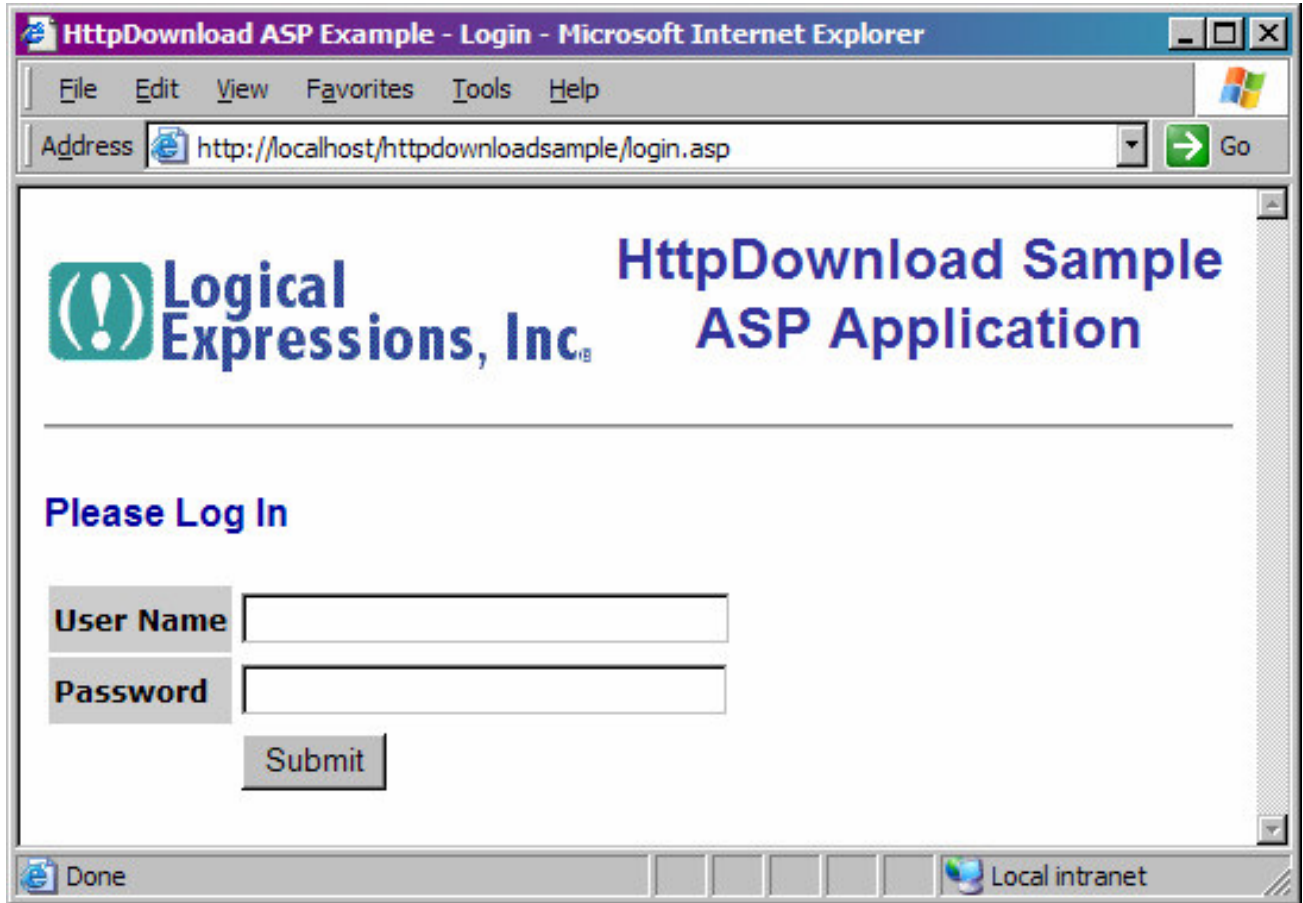
The MyDownloadFiles folder must allow read access by the anonymous Internet user account (typically IUSR_MachineName). That access is required at the Windows level in order for the HttpDownload class to read the files off disk. By the way, granting read access to the IUSR account does not make the files accessible through the Web site, it just makes them accessible to the HttpFileTransfer component which runs with IUSR credentials. Your host should be aware of this issue and anticipate your request to set the permissions up as described.

ASP Example

You can use the HttpDownload class from your ASP pages as you would any other COM class. The following example consists of three simple ASP pages. These pages demonstrate how you could password protect a series of download files.

Login.asp

The Login.asp page prompts the visitor for a username and password. If the information is correct, the user is redirected to the FileList.asp page. If validation fails, the user must try again.



The page defines two procedures: ValidateForm and DisplayForm. When users submit the form, the page executes ValidateForm. If validation is successful, the page redirects to FileList.asp.

Excerpt from Login.asp

```
Dim mstrUserName, mstrPassword, mstrUserMessage

If Len(Request.Form) > 0 Then
    If ValidateForm() Then
        Response.Redirect("FileList.asp")
    Else
        Call DisplayForm()
    End If
Else
    Call DisplayForm()
End If
```

The DisplayForm procedure is pretty straightforward, it just displays the login form along with any error message that may have been generated.

The ValidateForm procedure is more interesting because it authenticates the user's session.

ValidateForm procedure from Login.asp

```
Function ValidateForm()  
    Dim boolContinue  
  
    ' Clear any previous login info.  
    Session("Login.UserName") = ""  
  
    ' Get input from the submitted form:  
    mstrUserName = Request.Form("txtUserName")  
    mstrPassword = Request.Form("txtPassword")  
  
    ' In a real application, you would have a database lookup here or  
    ' something else more robust than hard-coded values.  
    boolContinue = (LCase(mstrUserName) = "bart")  
    If boolContinue Then  
        boolContinue = (mstrPassword = "CowaBunga!")  
    End If  
  
    If boolContinue Then  
        Session("Login.UserName") = mstrUserName  
    Else  
        mstrUserMessage = "Invalid user name or password. Please try again."  
    End If  
  
    ValidateForm = boolContinue  
End Function
```

For simplicity sake, the sample uses a Session variable to hold the login status of the visitor. This is not a particularly scalable approach for a production application, but it works well enough for the sample.

In each of the "protected" pages on the site, which includes the FileList page and the DownloadFile page, a small snippet of code at the top prevents unauthorized use:

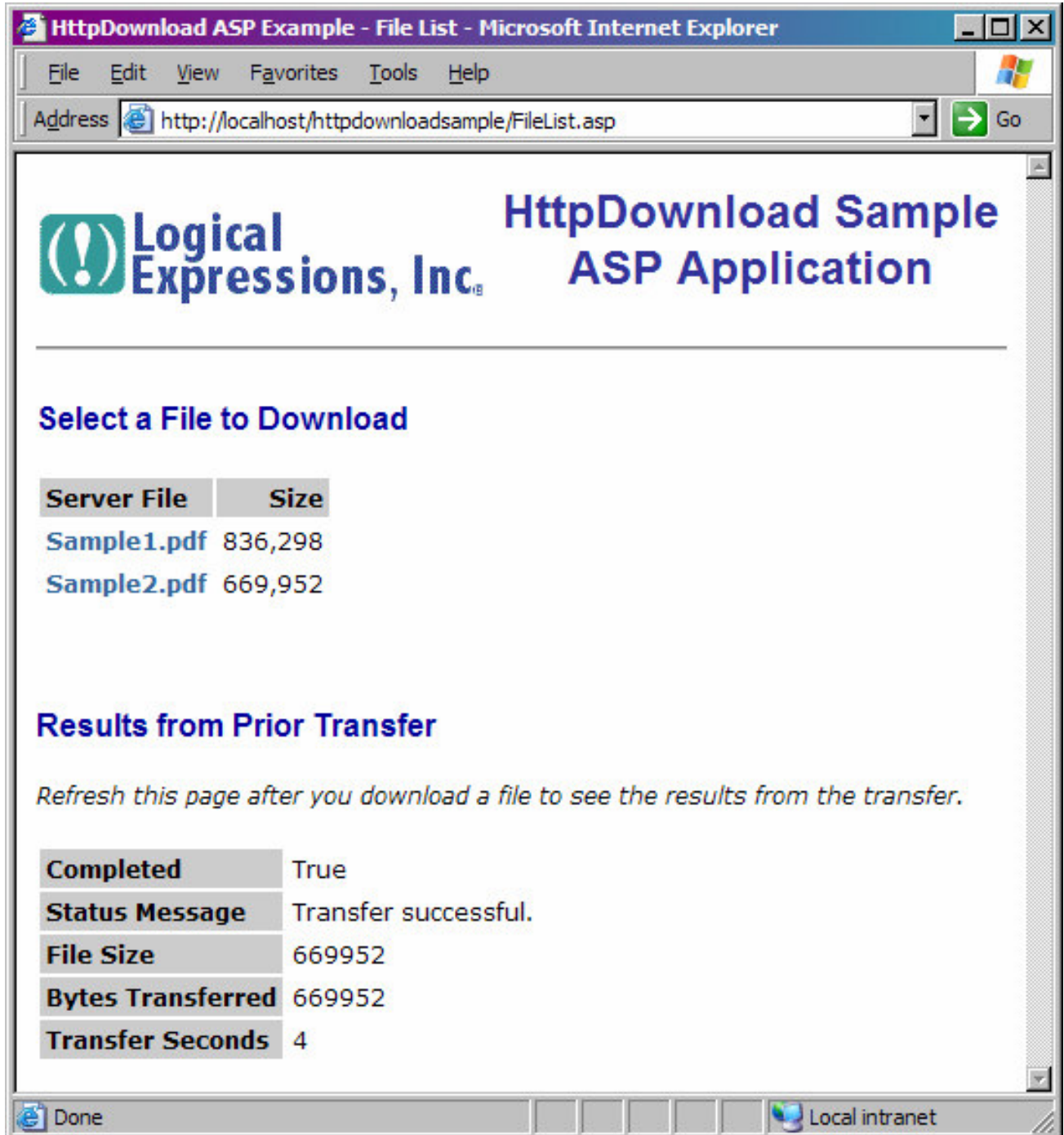
Login verification logic

```
' Redirect to login form if no user name has been written to the session.  
If Len(Session("Login.UserName")) = 0 Then  
    Response.Redirect("Login.asp")  
End If
```

The authentication process is very simple: If the user name has not been set in the session, redirect the request back to the login page.

FileList.asp

The FileList.asp file displays all of the files that are available for download along with their size. When you click on a file link, you invoke DownloadFile.asp, which is described below.



Although the file links invoke DownloadFile.asp, your browser does not actually navigate away from the FileList.asp page, so after the download you can hit the refresh button to see what happened

during the download. The refresh updates the "Results from Prior Transfer" section of the page, which shows you status information captured by DownloadFile.asp.

Rather than hard-coding the file links, FileList.asp builds them dynamically using the FileSystemObject, as shown below:

Dynamically building the list of files with FileSystemObject

```
<h2>Select a File to Download</h2>
<table border="0" cellspacing="2" cellpadding="3">
  <tr>
    <td class="Label">Server File</td>
    <td class="Label" align="right">Size</td>
  </tr>
<% Dim objFS, objFolder, objFile

Set objFS = Server.CreateObject("Scripting.FileSystemObject")
Set objFolder = objFS.GetFolder(mconDownloadFolder)

For Each objFile In objFolder.Files
%>
  <tr>
    <td><a href="DownloadFile.asp?File=<%= Server.URLEncode(objFile.Name) %>">
      <%= objFile.Name %></a></td>
    <td class="Normal" align="right"><%= FormatNumber(objFile.Size, 0) %></td>
  </tr>
<% Next
Set objFile = Nothing
Set objFolder = Nothing
Set objFS = Nothing
%>
</table>
```

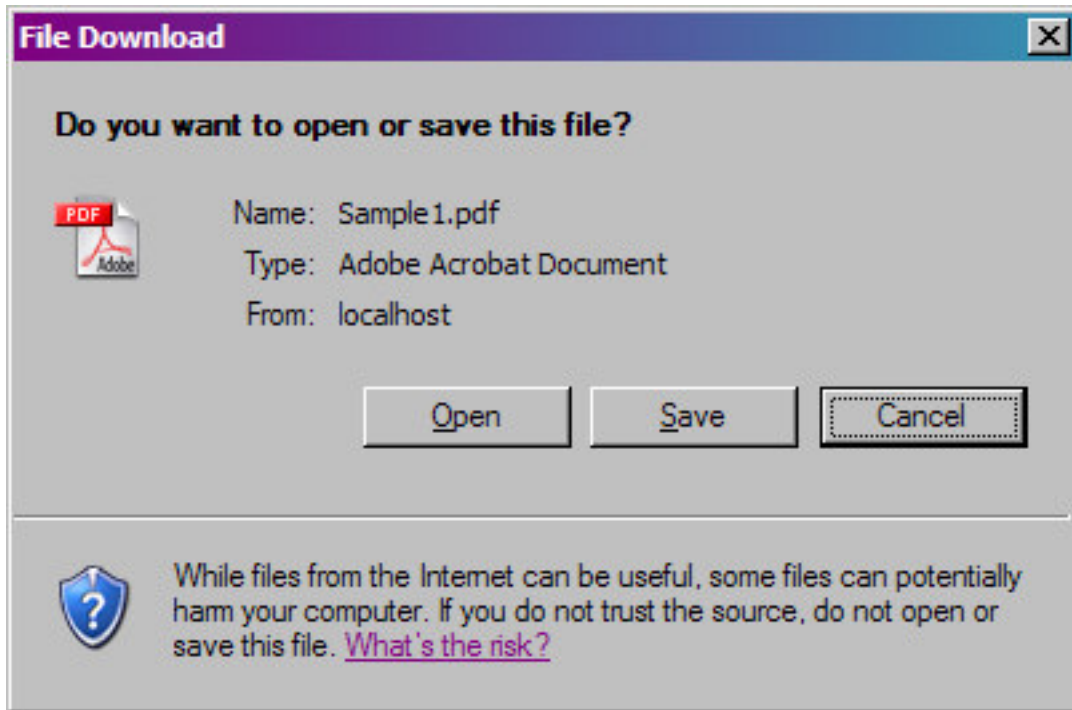
FileList.asp fills a 2-column table with file names and file sizes. The first step is to allocate an instance of the FileSystemObject and use it to retrieve information about the download folder. The download folder is identified by a constant (mconDownloadFolder) that is declared in Config.asp and included at the top of the file. Once it retrieves folder information, it gets the file name and size for each file in the folder's Files collection.

In a real application, you might have additional layers of security at this point. For example, you might restrict what files are available to each user. This example assumes that if you know the user name and password, you have access to all of the download files.

DownloadFile.asp

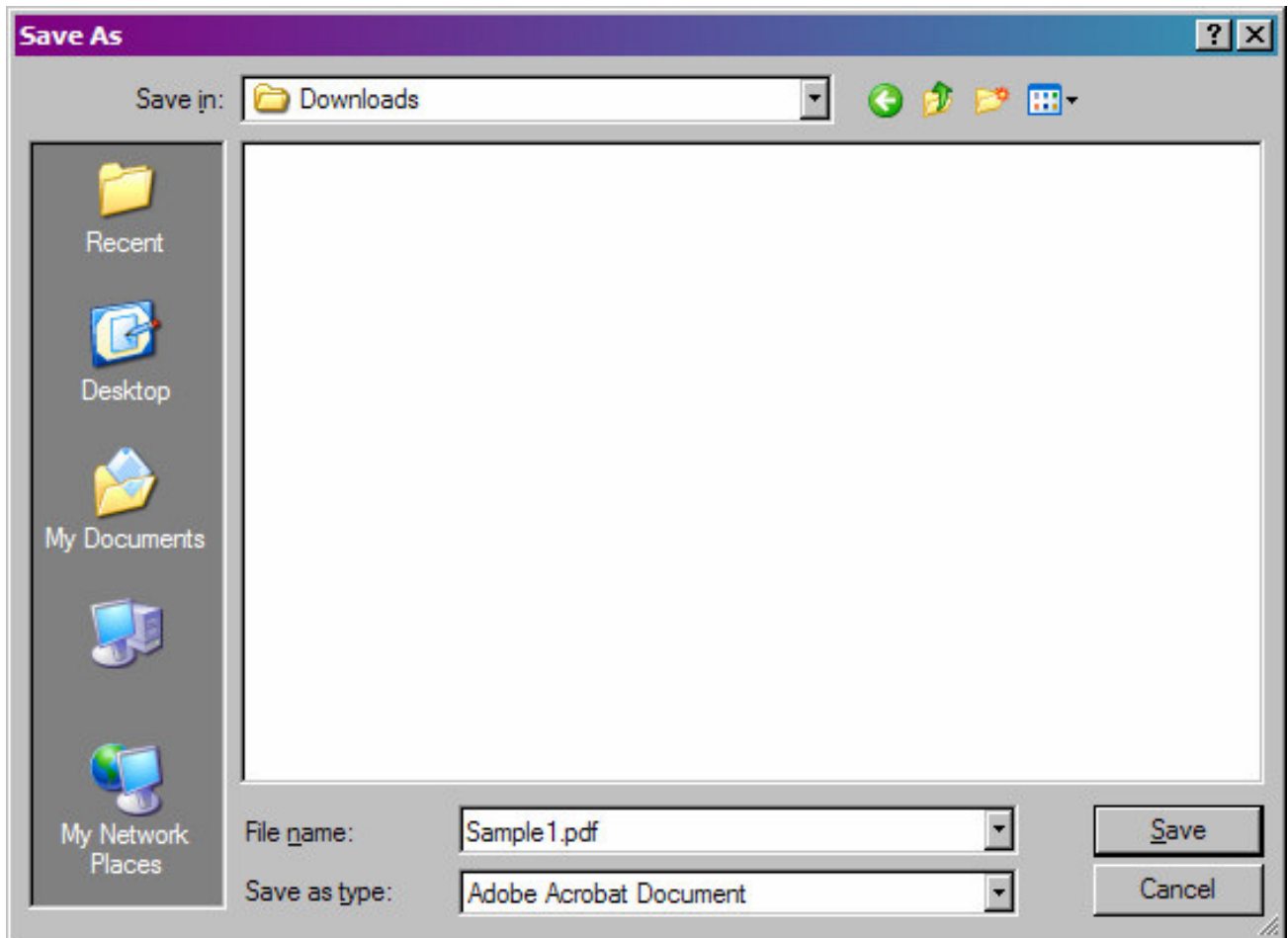
The DownloadFile.asp doesn't actually display an interface, which is why your browser does not navigate away from FileList.asp when you perform a download. Instead, it performs a binary transfer of data to the browser with special response headers that trigger your browser's Save As dialog.

The first thing you typically see is a security dialog that warns you about downloading files. An example from Internet Explorer is shown below.



Note how the browser has already figured out what kind of file you are trying to download. The special headers I mentioned earlier are responsible for passing that information.

Assuming you clicked Save on the File Download dialog, the next thing you should see is the Save As dialog. This dialog lets you select a location to store the downloaded file.



Once you hit Save on the Save As dialog, your browser completes the file transfer and stores the file in the selected location.

Now, the whole point of the sample application was to put the HttpDownload class to work, so naturally the code in DownloadFile.asp is the most interesting in that regard.

Calling HttpDownload to transfer a file

```
' Get the name of the requested file and assemble the source path:
strClientFileName = Request.QueryString("File")
strServerFilePath = mconDownloadFolder & "\" & strClientFileName
' Determine the MIME file type using the file extension:
Select Case LCase(Right(strClientFileName, 4))
  Case ".pdf"
    strMIMEContentType = "application/pdf"
  Case ".zip"
    strMIMEContentType = "application/x-zip-compressed"
  Case Else
    strMIMEContentType = "application/x-msdownload"
End Select

' Perform the file transfer:
Set objHttpDownload = Server.CreateObject("LEIHttpFileTransfer.HttpDownload")
Call objHttpDownload.Transfer(
  CStr(strServerFilePath), CStr(strMIMEContentType), CStr(strClientFileName))
' Preserve the results of the transfer:
Session("HttpDownload.TransferByteCount") = objHttpDownload.TransferByteCount
Session("HttpDownload.TransferCompleted") = objHttpDownload.TransferCompleted
Session("HttpDownload.TransferFileSize") = objHttpDownload.TransferFileSize
Session("HttpDownload.TransferSeconds") = objHttpDownload.TransferSeconds
```

```
Session("HttpDownload.TransferStatusMessage") = objHttpDownload.TransferStatusMessage
Set objHttpDownload = Nothing
```

The FileList page passes the requested file name down to the DownloadFile script through the File query string parameter. DownloadFile assembles a full file path by concatenating the file name to the download folder name.

DownloadFile figures out the correct MIME file type by looking at the file extension. For example, a .pdf file is expected to be an Adobe Acrobat file, which has a MIME file type of "application/pdf". The MIME file type tells your browser what kind of file is being passed to it, which helps it give you meaningful information in the File Download and Save As dialogs.

The actual work is accomplished by three lines of code at the end: The two Set statements and the Call statement. The lines that set session variables are just preserving the results of the call so the information can be reported on the FileList.asp page as described earlier.

The first Set statement allocates an instance of the HttpDownload class with Server.CreateObject. Then all you have to do is call that object's Transfer method, telling it what file to download, what MIME type to use, and what you want to call the file on the client computer. Alternatively, you can set the properties that correspond to those parameters and call Transfer without passing anything.

Note that the client file name and the source file name can be something completely different. This capability is handy if you need to give the files an unfriendly name to maintain uniqueness on the server, but want to give a friendly name to the client. You would need some kind of mechanism to map the names to one another, of course, but odds are good that your application will store meta data about your download files in a database, and that would be a great place to store the name mapping.

Visual Basic Example

If you use compiled Visual Basic components in your Web sites, you can still take advantage of the HttpDownload class. Because it is a COM component (also written in Visual Basic), it is easy to integrate with your Web application code.

Configuring Your VB Project

As with all COM components, the first thing you should do is set a reference to HttpFileTransfer in your Visual Basic project. The following instructions assume you have installed the HttpFileTransfer component on your development system.

- From the VB IDE, choose Project, References to display the References dialog box.
- Look for the entry named LEI HTTP File Transfer Component. Check the corresponding box and click OK to close the dialog box.

Using the HttpDownload Class

Once your project knows about the file transfer component, you can create instances of the `HttpDownload` class. Here is an example of a function that calls the component, taking in the name of the file to be transferred and the folder containing the download files:

TransferFileToClient Sample VB Function

```
Private Function TransferFileToClient(  
    ServerFileName As String,  
    FolderName As String,  
    ClientFileName As String,  
    ContextObject As ASPTypelibrary.ScriptingContext  
)  
    Dim strServerFilePath As String  
    Dim strMIMEContentType As String  
    Dim objDownload As LEIHttpFileTransfer.HttpDownload  
  
    strServerFilePath = FolderName & "\" & ClientFileName  
    ' Determine the MIME file type using the file extension:  
    Select Case LCase$(Right$(ClientFileName, 4))  
        Case ".pdf"  
            strMIMEContentType = "application/pdf"  
        Case ".zip"  
            strMIMEContentType = "application/x-zip-compressed"  
        Case Else  
            strMIMEContentType = "application/x-msdownload"  
    End Select  
  
    Set objDownload = New LEIHttpFileTransfer.HttpDownload  
    ' Pass the context object down to the component.  
    Call objDownload.OnStartPage(ContextObject)  
    Call objDownload.Transfer(  
        strServerFilePath, strMIMEContentType, ClientFileName)  
    TransferFileToClient = objDownload.TransferCompleted  
    Set objDownload = Nothing  
  
ProcExit:  
    Exit Function  
ProcError:  
    Resume ProcExit  
End Function
```

As you would expect, the code looks a lot like what you would do in ASP. The main difference is that you have to pass the ASP page context down to the `HttpDownload` object so it can write the server file to the ASP Response object. That implies that your VB code already did something to get the context itself. In the ASP example, the `Server.CreateObject` method takes care of this for you, because it automatically calls `OnStartPage` and passes the context when it loads the object. Note that it is not necessary to call `OnEndPage` because setting the object to nothing performs the necessary cleanup.

Obviously, your VB code must also establish the server file name, server folder path, and client file name before it calls the `TransferFileToClient` function. How that code works is up to you.

HttpDownload Class Reference

Use the HttpDownload class to transfer a file from the Web site server to the client browser. This section describes the properties and methods of the class.

Properties

Property	VB Type	Direction	Description
ServerFilePath	String	Input/Output	Gets/Sets the physical path on the server of the file to be transferred. IIS must have permissions to access the file.
MIMEContentType	String	Input/Output	Gets/Sets the MIME content type for the downloaded file. Common values are "application/pdf" and "application/x-zip-compressed".
ClientFileName	String	Input/Output	Gets/Sets the file name suggested by the Save As dialog.
TransferBufferSize	Long	Input/Output	Gets/Sets the file buffer size used by the Transfer method, which transmits the file in binary chunks. The default is 64K.
TransferCompleted	Boolean	Output	Gets the completion status of the transfer. True means the transfer was successful. If False, check TransferStatusMessage for an explanation of the failure. Valid only after a call to the Transfer method.
TransferFileSize	Long	Output	Gets the size of the file that was transferred. Valid only after a call to the Transfer method.
TransferByteCount	Long	Output	Gets the number of bytes transferred during the transfer operation. If the transfer completed successfully, this value should equal TransferFileSize. Valid only after a call to the Transfer method.
TransferSeconds	Long	Output	Gets the number of seconds the transfer took to complete. Note that this figure includes the time waiting for a response to the Save As dialog. Valid only after a call to the Transfer method.
TransferStatusMessage	String	Output	Gets an explanation of the transfer completion status. If TransferCompleted is True, the value is "Transfer successful." If False, the value provides the reason for the failure. Valid only after a call to the Transfer method.

Methods

Method	Parameters	Description
Transfer	Optional pstrServerFilePath As String Optional pstrMIMEContentType As String Optional pstrClientFileName As String	Use this method to initiate a file transfer from the server to the client.
OnStartPage	pobjContext As ASPTypeLibrary.ScriptingContext	Executed automatically by IIS when component reference is created with Server.CreateObject.
OnEndPage	(none)	Executed automatically by IIS when component reference is released.

Note that your ASP code should never need to call OnStartPage or OnEndPage. IIS handles that for you as long as you use Server.CreateObject to instantiate the HttpDownload object. You will need to call OnStartPage and pass the ASP scripting context if you instantiate the HttpDownload object using "early binding" (using Dim/As New or Dim/Set New) from a compiled Visual Basic module. See the Visual Basic example for more information about this.

End of Document