CHAPTER 33

Interacting with the Operating System

IN THIS CHAPTER

ColdFusion provides the developer with many tools with which to interact with the operating system. These tools include functions and tags to manipulate files and directories using <CFFILE> and <CFDIRECTORY>, execute applications on the server using the <CFEXECUTE> tag, and manipulate the system Registry using the <CFREGISTRY> tag. This chapter shows how these tags can be used to interact with the file system and operating system.

Introduction to <CFFILE>

<CFFILE> permits local file access through CFML templates. Files can be moved, copied, renamed, or deleted by using various action attributes for the <CFFILE> tag. Additionally, <CFFILE> provides mechanisms for reading and writing ASCII files with ColdFusion. Taking advantage of the <CFFILE> tag provides you with the ability to produce complex applications with file manipulation using a single interface. The templates in which the <CFFILE> tag is used can be protected using native server security when the templates are stored in directories below the document root defined for the HTTP server. In addition to the ability to access the local file system, <CFFILE> provides the ability to upload files using the HTTP protocol.

The Varied Faces of <CFFILE>

The <CFFILE> tag performs different operations depending on the value of its ACTION attribute.

For moving, copying, or renaming files on the server's drives, it looks like this:

```
<CFFILE
  ACTION="Copy or Move or Rename"
  SOURCE="c:\LocationOnServer\MySourceFile.txt"
  DESTINATION="c:\AnotherLocationOnServer\MyNewFile.txt">
```

For deleting a file on the server's drive, it looks like this:

```
<CFFILE
  ACTION="Delete"
  FILE="c:\LocationOnServer\MyFileToDelete.txt">
```

For creating or adding to existing files on the server's drives, it looks like this:

```
<CFFILE
  ACTION="Write or Append"
  FILE="c:\LocationOnServer\MyFile.txt"
  OUTPUT="#ContentToSaveInFile#">
```

For reading a file on the server on the server's drive, it looks like this:

```
<CFFILE
  ACTION="Read or ReadBinary"
  FILE="c:\LocationOnServer\MyFile.txt"
  VARIABLE="VariableNameToHoldContent">
```

Finally, for uploading a file from the browser machine to the server, it looks like this:

```
<CFFILE
  ACTION="Upload"
  FILEFIELD="MyFormInput"
  DESTINATION="c:\LocationOnServer">
```

NOTE

Because ColdFusion operates on the server, it has no direct access to the client file system, so it can't read, copy, delete, or do anything else to the files on the browser machine. The only file-related thing it can do is to accept file uploads from the browser, which must be explicitly initiated by the user. Keep this in mind when developing your applications.

As you can see, the `<CFFILE>` tag's attributes can be set to various values depending on the task at hand. Of course, each of the attributes can be set dynamically using variables created via the `<CFSET>` tag or with the values of query or form fields. (When using form fields, extreme care should be taken to ensure that security restrictions are in place to prevent malicious action as a result of dynamic file action.) Table 33.1 indicates the attributes and the valid values permitted for specific values of the `ACTION` attribute.

NOTE

When using `FORM` fields, `URL` variables, or other user-entered data to set the attributes for the `<CFFILE>` tag, extreme caution should be used to ensure only valid entries are processed.

Table 33.1 `<CFFILE>` Tag `ACTION` Attributes

ACTION	DESCRIPTION
`COPY`	Copies a file from the location specified in `SOURCE` to the location specified in `DESTINATION`.
`MOVE`	Moves a file from the location specified by `SOURCE` to the location specified in `DESTINATION`.
`DELETE`	Deletes the file specified by the `FILE` attribute.
`RENAME`	Renames the file specified in `SOURCE`, giving it the new name specified in `DESTINATION`.

Table 33.1 (continued)

ACTION	DESCRIPTION
READ	Reads the contents of the text file specified by FILE to into the string variable specified by VARIABLE.
READBINARY	Reads the contents of the binary file specified by FILE into a binary object variable specified by VARIABLE.
WRITE	Writes the contents of the string specified in OUTPUT to the file specified by FILE. If the file already exists, the existing file is completely replaced by the new one.
APPEND	Writes the contents of the string specified in OUTPUT to the file specified by FILE. If the file already exists, the new content is appended to the end of the existing content.
UPLOAD	Used to upload files. Accepts a file from the browser machine and saves it to the location on the server specified in DESTINATION. The FILEFIELD attribute must correspond to the name of an <INPUT> form field of TYPE="File". The NAMECONFLICT attribute controls what happens a file with the same name already exists on the server.

NOTE

For all the actions that create files (Write, Append, Move, Copy, Rename, and Upload), you can also specify an ATTRIBUTES attribute to control the file's attributes on disk. For instance, when using ACTION="Write" to create a new file, you could use ATTRIBUTES="ReadOnly" to make the new file be considered read-only. For details, see Appendix B, ColdFusion Tag Reference.

NOTE

For UNIX servers, you can also provide a MODE attribute for actions that create completely new files (Write, Append, and Upload). This attribute gives you a way to control the chmod-style values for files. For details, see Appendix B, ColdFusion Tag Reference.

Accessing the Server's Filesystem

During application development you might need to perform local filesystem operations (local here refers to the Web server's file system). This need might manifest itself in the requirement to read or write ASCII files or to copy, move, rename, or delete various application files.

Reading and Writing Files

Using <CFFILE> to read and write ASCII files is fairly straightforward. For example, to read the contents of the README.txt file in the C:\CFusionMX\runtime\jre folder, you could use code such as Listing 33.1.

Listing 33.1 SimpleFileRead.cfm—<CFFILE> usage for reading README.txt into a variable

```
<!---
  Filename:  SimpleFileRead.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Exhibits how to read and display the contents of a text file
--->
```

Listing 33.1 (CONTINUED)

```
<!--- Read the contents of the text file into a string variable --->
<CFFILE
  ACTION="Read"
  FILE="C:\CFusionMX\runtime\jre\README.txt"
  VARIABLE="ReadmeContent">

<!--- Display the value --->
<CFOUTPUT>
  <TABLE BORDER="0" CELLPADDING="5" CELLSPACING="0">
    <TR>
      <TD STYLE="background:navy;color:white;font-weight:bold">
        The first 1000 characters of the README.txt file are:
      </TD>
    </TR><TR>
      <TD BGCOLOR="Silver">
        #HTMLCodeFormat(Left(ReadmeContent, 1000))#
      </TD>
    </TR>
  </TABLE>
</CFOUTPUT>
```

NOTE

<CFFILE> can read both ASCII and binary files. To read a binary file, the ACTION attribute must be set to ReadBinary.

This is a simple yet powerful feature. The example in Listing 33.1 is trivial (it simply reads the contents of the README file on the server's drive, then displays the first 1000 characters on a web page), but it serves as the basis for the power of reading files using <CFFILE>. After the <CFFILE> operation is completed, the contents of the file are available in the variable specified during the call (in this case, ReadmeContent). If this file contained delimited data, it could be parsed using <CFLOOP> and various string functions.

Writing a file using <CFFILE> is just as easy. Listing 33.2 shows an example of writing a modified version of the CFDIST.INI file back out to disk.

Listing 33.2 SimpleFileWrite.cfm—<CFFILE> Usage to Alter the Contents of a Text File

```
<!---
  Filename: SimpleFileWrite.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Exhibits how to read, change, and re-write a text file
--->

<!--- Read the contents of the text file into a string variable --->
<CFFILE
  ACTION="Read"
  FILE="C:\CFusionMX\runtime\jre\README.txt"
  VARIABLE="ReadmeContent">

<!--- Modify the contents of the variable --->
<CFSET header = "File Modified using ColdFusion MX on: ">
<CFSET header = header & DateFormat(Now(),"mm/dd/yyyy") & " at ">
<CFSET header = header & TimeFormat(Now(),"h:mm:ss tt") & Chr(13) & Chr(10)>
<CFSET RevisedContent = header & ReadmeContent>
```

Listing 33.2 (continued)

```
<!--- Write the contents of the variable back out to disk --->
<CFFILE
  ACTION="WRITE"
  FILE="C:\CFusionMX\runtime\jre\README.txt"
  OUTPUT="#RevisedContent#"
  ADDNEWLINE="Yes">

<HTML>
<HEAD>
  <TITLE>&lt;CFFILE&gt; read/write Example</TITLE>
</HEAD>
<BODY>

<!--- Display the file's revised contents --->
<CFOUTPUT>
  <TABLE BORDER="0" CELLPADDING="5" CELLSPACING="0">
    <TR>
      <TD STYLE="background:navy;color:white;font-weight:bold">
        C:\CFusionMX\runtime\jre\README.txt was modified, as shown below:
      </TD>
    </TR><TR>
      <TD BGCOLOR="Silver">
        #HTMLCodeFormat(RevisedContent)#
      </TD>
    </TR>
  </TABLE>
</CFOUTPUT>

</BODY>
</HTML>
```

The code in Listing 33.2 builds on the example in Listing 33.1. First, it uses `<CFFILE>` to read the contents of `CFDIST.INI` into a variable. Next, a new line is added to the beginning of the variable by concatenating a remark statement coupled with a date/time stamp to the contents of the variable. Lastly, `<CFFILE>` is called again to write the contents of the variable back out to disk. The resulting file output is displayed as it would be seen on disk.

NOTE

`<CFFILE>` with the `ACTION` attribute set to `Write` creates the file if it does not exist and overwrites the file if it exists. Care should be taken to ensure that existing content is not deleted inadvertently. If the contents of an existing ASCII file are to be kept, `<CFFILE>` should be used with the `ACTION` set to `Append`, which will concatenate the contents of the variable specified in the `OUTPUT` attribute to the end of the disk file. The `FileExists()` function can be used to determine whether a `Write` or `Append` operation should take place.

Copying, Moving, Renaming, and Deleting Files

The `<CFFILE>` tag provides the capability to perform local file operations, such as `COPY`, `MOVE`, `RENAME`, and `DELETE`. Local in this example means local to the ColdFusion server—not local to the browser machine. These actions have the potential for causing severe damage to the filesystem. Security considerations should therefore be evaluated carefully before developing ColdFusion templates that provide the ability to copy, rename, move, or delete files.

NOTE

Security measures can vary by operating system and from one Web server to another. Consult documentation specific to the configuration of your Web server for detailed information about security issues.

To provide local file access, the <CFFILE> tag is used with the ACTION attribute set to COPY, MOVE, RENAME, or DELETE. The DESTINATION attribute is not required in the case of the DELETE action value; it is required in all other cases.

Listing 33.3 shows ColdFusion's capability to copy files on the local filesystem. The ACTION attribute is set to COPY; the SOURCE attribute is set to the name of the file that is to be copied. The DESTINATION attribute is set to the directory into which the file will be copied. The DESTINATION attribute also can specify a filename in addition to the directory name, which enables you to copy one file to another while changing the name in the process.

Listing 33.3 SimpleFileCopy.cfm—<CFFILE> Tag with ACTION Attribute Set to COPY

```
<!---
  Filename: SimpleFileCopy.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Exhibits how to make a copy of a file on the server's drive
--->


<!--- Copy a file from one location to another --->
<CFFILE
  ACTION="Copy"
  SOURCE="C:\CFusionMX\runtime\jre\README.txt"
  DESTINATION="C:\Inetpub\wwwroot\ows\README.txt">

<CFOUTPUT>The file has been copied.</CFOUTPUT>
```

Listing 33.4 shows ColdFusion's capability to move files on the local filesystem. The ACTION attribute is set to MOVE; the SOURCE attribute is set to the name of the file that is to be moved. The DESTINATION attribute is set to the directory into which the file will be moved.

Listing 33.4 SimpleFileMove.cfm—<CFFILE> Tag with ACTION Attribute Set to MOVE

```
<!---
  Filename:  SimpleFileMove.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Exhibits how to move a file from one
             location to another on the server's drives
--->

<!--- Move a file from one location to another --->
<CFFILE
  ACTION="Move"
  SOURCE="C:\CFusionMX\runtime\jre\README.txt"
  DESTINATION="C:\Inetpub\wwwroot\ows\README.txt">

<CFOUTPUT>The file has been moved.</CFOUTPUT>
```

Listing 33.5 shows the use of the DELETE value of the ACTION attribute. The ACTION attribute is set to DELETE, and the FILE attribute is set to the name of the file you want deleted.

Listing 33.5 SimpleFileDelete.cfm—<CFFILE> Tag with ACTION Attribute Set to DELETE

```
<!---
  Filename: SimpleFileDelete.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Exhibits how to make a remove a file from the server's drive
--->

<!--- Delete a file from the server's drive --->
<CFFILE
  ACTION="Delete"
  FILE="C:\Inetpub\wwwroot\ows\README.txt">

<CFOUTPUT>The file has been deleted.</CFOUTPUT>
```

NOTE

Use the DELETE action carefully. Access to templates that delete files should be carefully restricted.

Listing 33.6 shows <CFFILE> being used to RENAME an existing file.

Listing 33.6 SimpleFileRename.cfm—<CFFILE> Tag with ACTION Attribute Set to RENAME

```
<!---
  Filename:  SimpleFileMove.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Exhibits how to rename a file on the server's drive
--->

<!--- Rename a file on the server's drive --->
<CFFILE
  ACTION="Rename"
  SOURCE="C:\Inetpub\wwwroot\ows\README.txt"
  DESTINATION="C:\Inetpub\wwwroot\ows\ReadMeAgain.txt">

<CFOUTPUT>The file has been renamed.</CFOUTPUT>
```

Uploading Files

Browser-based file uploads in ColdFusion are provided through the <CFFILE> tag. This tag takes advantage of features available in most Web browsers that support file uploads using the HTTP protocol. The syntax of the <CFFILE> tag can be used with selected attributes to facilitate the uploading of files to the server.

NOTE

The method by which files are uploaded to the server using HTTP is documented in the Internet Request for Comment (RFC) 1867, which was available at http://www.faqs.org/rfcs/rfc1867.html at the time of this writing. RFC 1867 is the formal documentation of the HTTP file upload process. It specifies the concepts related to file uploads using MIME file extensions.

NOTE

Netscape browsers have supported file uploading since version 2.0. Internet Explorer has supported it since version 4.0 (it was also possible in IE 3.02 with a separate add-on). If your users have other browsers, you might want to double-check to make sure that file uploading is supported.

To upload a file from the browser machine to the server, the <CFFILE> tag is used like this:

```
<CFFILE
  ACTION="Upload"
  FILEFIELD="MyFormInput"
  DESTINATION="c:\LocationOnServer"
  NAMECONFLICT="OVERWRITE"
  ACCEPT="image/gif">
```

Briefly, the meaning of each of these attributes is as follows:

- The FILEFIELD attribute must correspond to a special <INPUT> field on a HTML form (you will see an example of this shortly, in Listing 33.7).
- The DESTINATION is the folder on the server that you would like the file to be placed into when the upload is complete.
- The NAMECONFLICT attribute controls what happens if there is already a file in the destination folder that has the same name as the file being uploaded.
- The ACCEPT attribute allows you to control which types of files that the user is able to upload (just images, just text files, and so on).

You must carefully examine a number of issues prior to writing the HTML/CFML necessary to process a file upload. First and foremost is security. The directory to which the files will be uploaded must be secure from outside view, and the templates used to perform the file operations must be protected from unauthorized access. Because the threat of computer viruses is increasing, you must take precautions to protect your system from malicious users. The second issue to examine is the reason you are providing file operations to the users. Is it necessary? Can it be accomplished using other means?

What If the File Already Exists on the Server?

It's often important to be able for your server to be able to accept file uploads from multiple users at the same time. If the users are uploading files with the same filenames (or filenames that already exist on the server from prior uploads), you need to tell ColdFusion how to handle the situation using the NAMECONFLICT attribute. Table 33.2 lists the values you can supply to the NAMECONFLICT attribute.

Table 33.2 NAMECONFLICT Values for <CFFILE> ACTION="Upload"

VALUE	MEANING
NAMECONFLICT="Error"	If a file with the same name already exists on the server, an error message is generated and page execution stops. Of course, you can catch and recover from the error using the <CFTRY> and <CFCATCH> tags (discussed in Chapter 31, Error Handling).

Table 33.2 (continued)

VALUE	MEANING
`NAMECONFLICT="Skip"`	If a file with the same name already exists on the server, the file upload operation is simply skipped. No error message is shown. Your code can examine the value of the `CFFILE.FileWasSaved` variable to detect whether the upload was actually skipped for this reason (see Table 33.3).
`NAMECONFLICT="Overwrite"`	If a file with the same name already exists on the server, the existing file is overwritten with the file being uploaded from the browser. Your code can look at the value of `CFFILE.FileWasOverwritten` to determine whether a file was actually overwritten when your code is actually used.
`NAMECONFLICT="MakeUnique"`	If a file with the same name already exists on the server, the file from the browser is saved with an automatically generated filename. This ensures that an uploaded file can always be saved without overwriting an existing file. Your code can use the `CFFILE.ServerFile` variable to determine the actual filename used (see Table 33.3).

Determining the Status of a File Upload

After a `<CFFILE>` operation is completed, information about the file is available in reference keys of the `CFFILE` structure. Similar to the `URL`, `FORM`, and `CGI` structures, the `CFFILE` structure maintains status information about the most recent file operation completed or attempted. Keys in the `CFFILE` structure are referenced in the same manner as other ColdFusion variables (for example, `#CFFILE.ContentType#`). Table 33.3 identifies the attributes maintained and their meanings.

NOTE

Hopefully this isn't too confusing, but previous versions of ColdFusion used a prefix of `FILE` for these variables, rather than `CFFILE`. Either prefix will work in ColdFusion MX to maintain backward compatibility, but you should use `CFFILE` for new applications.

Table 33.3 `CFFILE` Variables Available After a File Upload

KEY	EXPLANATION
`CFFILE.AttemptedServerFile`	Did ColdFusion attempt to save the file? (Yes/No)
`CFFILE.ClientDirectory`	Client-side directory in which the file was located.
`CFFILE.ClientFile`	Client-side filename (with extension).
`CFFILE.ClientFileExt`	Client-side filename extension without the period.
`CFFILE.ClientFileName`	Client-side filename (without extension).
`CFFILE.ContentSubType`	MIME content subtype of file.
`CFFILE.ContentType`	MIME content type of file.
`CFFILE.DateLastAccessed`	Returns the date and time the uploaded file was last accessed.

Table 33.3 (CONTINUED)

KEY	EXPLANATION
CFFILE.FileExisted	Did a file with the same name exist in the specified destination prior to upload, copy, or move? (Yes/No)
CFFILE.FileSize	Size of the uploaded file.
CFFILE.FileWasAppended	Was the file appended to an existing file by ColdFusion? (Yes/No)
CFFILE.FileWasOverwritten	Was an existing file overwritten by ColdFusion? (Yes/No)
CFFILE.FileWasRenamed	Was the uploaded file renamed to avoid a conflict? (Yes/No)
CFFILE.FileWasSaved	Was the file saved by ColdFusion? (Yes/No)
CFFILE.OldFileSize	Size of the file that was overwritten during an upload operation.
CFFILE.ServerDirectory	Directory on server where file was saved.
CFFILE.ServerFile	Filename of the saved file.
CFFILE.ServerFileExt	Extension of the uploaded file without the period.
CFFILE.ServerFileName	Filename without extension of the uploaded file.
CFFILE.TimeCreated	Returns the time the uploaded file was created.
CFFILE.TimeLastModified	Returns the date and time of the last modification to the uploaded file.

These variables are used in several of the examples that follow, specifically in Listing 35.8 and the examples that follow it.

Building an Upload Interface

After you have decided to use <CFFILE> to upload a file, you can move on to the next step of the process, which is preparing the user interface. This requires the development of an HTML form, either through writing static HTML or by creating an HTML form using dynamic code generated via CFML. In either case, the form's structure is basically the same.

The next series of listings is used to create an add-on to the actor listings that will allow a photo to be linked to an actor record. First, the general syntax is shown, and then specific modifications to the actor templates are made.

Listing 33.7 shows the HTML code necessary to create a form that prompts the user for a file to be uploaded to the server (Figure 33.1). The user can use the Browse button to select a file from a file selection dialog box (Figure 33.2); the selected filename will be placed in the TYPE="File" input field (Figure 33.3), which will cause the file to be uploaded when the form is submitted.

Figure 33.1

Example HTML form for file upload.

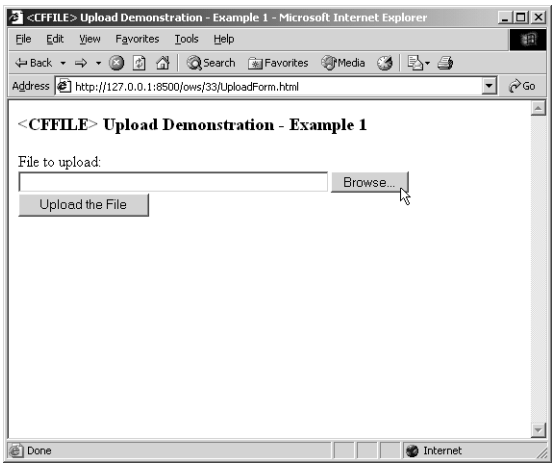


Figure 33.2

Example file selection dialog box.

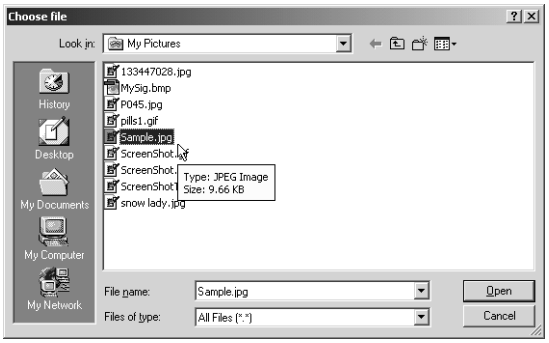
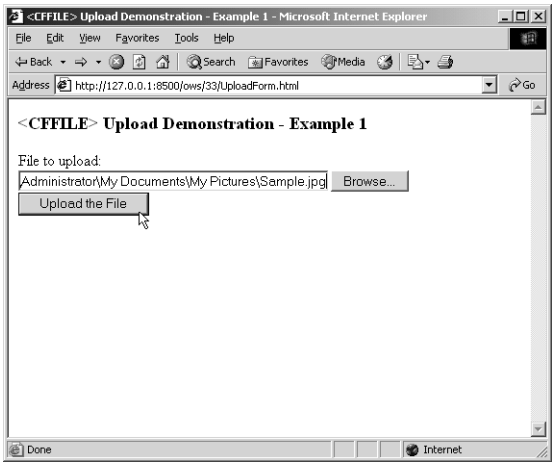


Figure 33.3

Example HTML form for file upload with selected filename.



Listing 33.7 UploadForm.html—HTML Form for File Upload Using the <CFFILE> Tag

```
<!---
  Filename:  UploadForm.html
  Edited By: Nate Weiss (NMW)
  Purpose:   Simple file uploading example
--->


<HTML>
<HEAD>
  <TITLE>&lt;CFFILE&gt; Upload Demonstration - Example 1</TITLE>
</HEAD>
<BODY>

<H3>&lt;CFFILE&gt; Upload Demonstration - Example 1</H3>


<!--- Create HTML form to upload a file --->
<FORM
  ACTION="UploadAction.cfm"
  ENCTYPE="multipart/form-data"
  METHOD="Post">

  <!--- File field for user to select or specify a filename --->
  <P>File to upload:<BR>
  <INPUT
    TYPE="File"
    NAME="FileName"
    SIZE="50"><BR>

  <!--- Submit button to submit the form (and upload the file) --->
  <INPUT
    TYPE="Submit"
    VALUE="Upload the File">
</FORM>

</BODY>
</HTML>
```

There are several important items within this form, all of which are necessary to perform a file upload:

The <FORM> tag has a ENCTYPE="multipart/form-data" attribute, which is necessary for the browser to send the file to ColdFusion in a way that it can use.

The addition of an <INPUT> of TYPE="File", which tells the browser to process file selection using the standard user-interface functionality of the underlying operating system.

The <FORM> tag's ACTION attribute identifies which ColdFusion template will be used to process the file. That template will use the <CFFILE> tag with ACTION="Upload".

The METHOD attribute is set to Post.

The dialog box shown in Figure 33.2 is specific to the operating system on which a browser is running and changes from one operating system to another. Figure 33.3 shows the HTML form with the text box filled with the selected filename.

When this form is submitted, the FORM tag's ACTION attribute causes the selected file to be uploaded. Listing 33.8 shows the CFML code required to process the uploaded file. This example enumerates the values of the keys in the CFFILE structure after the file has been written to the file server. Details of the keys in the CFFILE structure can be seen in Table 33.3.

Listing 33.8 UploadAction.cfm—Processing an Uploaded File with ColdFusion

```
<!---
  Filename:  UploadAction.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Demonstrates how to accept a file upload from the browser machine
--->

<!--- Template to process uploaded files from user --->
<HTML>
<HEAD>
  <TITLE>&lt;CFFILE&gt; Upload Demonstration - Example 1</TITLE>
</HEAD>
<BODY>

<H3>&lt;CFFILE&gt; Upload Demonstration - Example 1</H3>

<!--- Accept the actual file upload --->
<!--- The file will be placed into the same folder as this ColdFusion page --->
<CFFILE
  DESTINATION="#GetDirectoryFromPath(GetBaseTemplatePath())#"
  ACTION="Upload"
  NAMECONFLICT="Overwrite"
  FILEFIELD="FileName">


<!--- Output information about the status of the upload --->
<CFOUTPUT>
  <P>
    &lt;CFFILE&gt; Tag File Upload Demonstration Results - Example 1<br>
    File Upload was Successful! Information about the file is detailed below
  </P>

  <TABLE>
    <CAPTION><B>File Information</B></CAPTION>
    <TR VALIGN="Top">
      <TH ALIGN="Left">File Name:</TH>
      <TD>#File.ServerDirectory#\#CFFILE.ServerFile#</TD>
      <TH ALIGN="Left">Content Type:</TH><TD>#File.ContentType#</TD>
    </TR>
    <TR VALIGN="Top">
      <TH ALIGN="Left">Content SubType:</TH>
      <TD>#File.ContentSubType#</TD>
      <TH ALIGN="Left">Client Path:</TH>
      <TD>#File.ClientDirectory#</TD>
    </TR>
    <TR VALIGN="Top">
      <TH ALIGN="Left">Client File:</TH>
      <TD>#File.ClientFile#</TD>
      <TH ALIGN="Left">Client FileName:</TH>
      <TD>#File.ClientFileName#</TD>
    </TR>
    <TR VALIGN="Top">
      <TH ALIGN="Left">Client FileExt:</TH>
```

Listing 33.8 (CONTINUED)

```
          <TD>#File.ClientFileExt#</TD>
          <TH ALIGN="Left">Server Path:</TH>
          <TD>#File.ServerDirectory#</TD>
        </TR>
        <TR VALIGN="Top">
          <TH ALIGN="Left">Server File:</TH>
          <TD>#File.ServerFile#</TD>
          <TH ALIGN="Left">Server FileName:</TH>
          <TD>#File.ServerFileName#</TD>
        </TR>
        <TR VALIGN="Top">
          <TH ALIGN="Left">Server FileExt:</TH>
          <TD ALIGN="Left">#File.ServerFileExt#</TD>
          <TH ALIGN="Left">Attempted ServerFile:</TH>
          <TD>#File.AttemptedServerFile#</TD>
        </TR>
        <TR VALIGN="Top">
          <TH ALIGN="Left">File Existed?</TH>
          <TD>#File.FileExisted#</TD>
          <TH ALIGN="Left">File Was Saved?</TH>
          <TD>#File.FileWasSaved#</TD>
        </TR>
        <TR VALIGN="Top">
          <TH ALIGN="Left">File Was Overwritten?</TH>
          <TD>#File.FileWasOverWritten#</TD>
          <TH ALIGN="Left">File Was Appended?</TH>
          <TD>#File.FileWasAppended#</TD>
        </TR>
        <TR VALIGN="Top">
          <TH ALIGN="Left">File Was Renamed?</TH>
          <TD>#File.FileWasRenamed#</TD>
          <TH ALIGN="Left">File Size:</TH>
          <TD>#File.Filesize#</TD></TH>
        </TR>
        <TR VALIGN="Top">
          <TH ALIGN="Left">Old File Size:</TH>
          <TD>#File.OldFileSize#</TD>
          <TH ALIGN="Left">Date Last Accessed:</TH>
          <TD>#DateFormat(File.DateLastAccessed,'dd mmm yyyy')#</TD>
        </TR>
        <TR VALIGN="Top">
          <TH ALIGN="Left">Date/Time Created:</TH>
          <TD>
            #DateFormat(File.TimeCreated,'dd mmm yyyy')#
            #Timeformat(File.TimeCreated,'hh:mm:ss')#
          </TD>
          <TH ALIGN="Left">Date/Time Modified:</TH>
          <TD>
            #DateFormat(File.TimeLastModified,'dd mmm yyyy')#
            #Timeformat(File.TimeLastModified,'hh:mm:ss')#
          </TD>
        </TR>
      </TABLE>
    </CFOUTPUT>

    </BODY>
    </HTML>
```

The CFML template shown in Listing 33.8 processes the uploaded file, stores it in the directory indicated in the <CFFILE> tag's DESTINATION attribute, and then prints out the contents of the keys in the CFFILE structure (Figure 33.4). Some of the CFFILE keys might not have values, depending on the attributes passed to the <CFFILE> tag.

NOTE

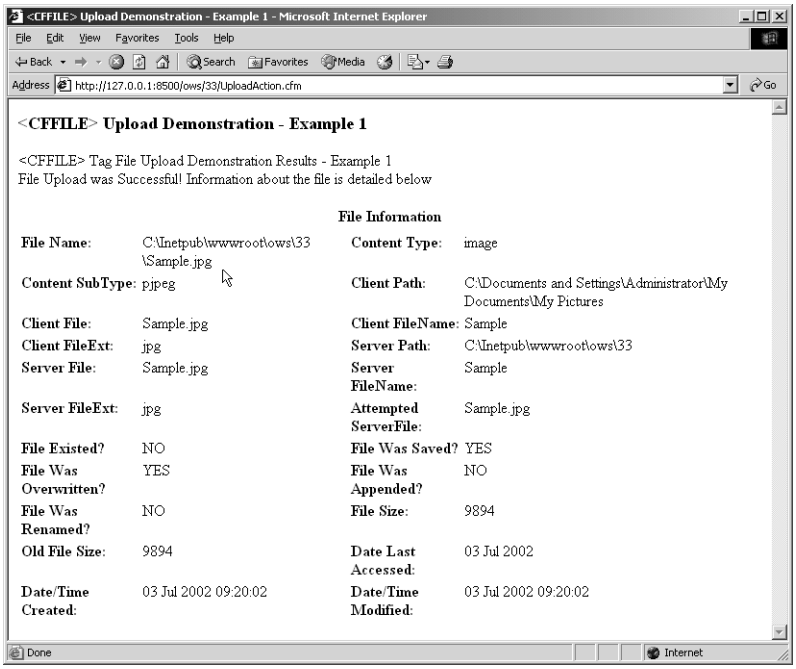
Listing 33.8 uses the GetDirectoryFromPath() and GetBaseTemplatePath() functions to set the DESTINATION attribute to the directory portion of the currently-executing template's filename. In other words, the uploaded file will be saved in the same folder that Listing 33.8 itself is stored in (probably the ows/32 folder within your web server's document root). This combination of functions can be used anytime For details about these helpful functions, see Appendix C, "ColdFusion Function Reference".

NOTE

If you wish, you could use #ExpandPath('./')# instead of #GetDirectoryFromPath(GetBaseTemplatePath())# to return the location of the currently-executing template. See Appendix C for details.

Figure 33.4

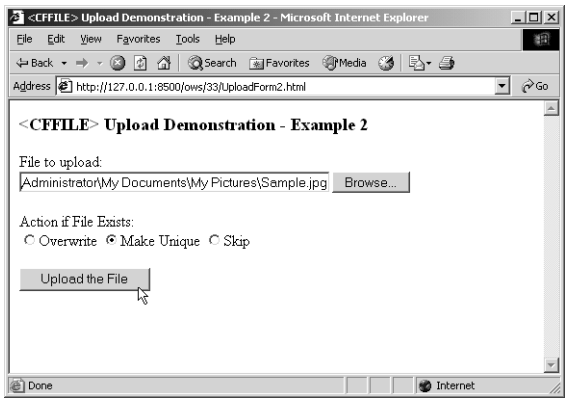
Example CFML output of uploaded file information.



Listing 33.9 shows an example that builds on the HTML/CFML code you just wrote; it demonstrates the use of variables to set the various attributes of the <CFFILE> tag. The HTML form has been modified by adding a radio button group that corresponds to the NAMECONFLICT attribute in the <CFFILE> tag (Figure 33.5).

Figure 33.5

Providing further control over file uploads with form fields.



Listing 33.9 `UploadForm2.html`—Modification of HTML to Demonstrate Data-Driven Attribute Setting

```
<!---
  Filename:  UploadForm2.html
  Edited By: Nate Weiss (NMW)
  Purpose:   Simple file uploading example
--->


<HTML>
<HEAD>
  <TITLE>&lt;CFFILE&gt; Upload Demonstration - Example 2</TITLE>
</HEAD>
<BODY>

<H3>&lt;CFFILE&gt; Upload Demonstration - Example 2</H3>


<!--- Create HTML form to upload a file --->
<FORM
  ACTION="UploadAction2.cfm"
  ENCTYPE="multipart/form-data"
  METHOD="Post">

  <!--- File field for user to select or specify a filename --->
  <P>File to upload:<BR>
  <INPUT
    TYPE="File"
    NAME="FileName"
    SIZE="50"><BR>

  <P>Action if File Exists:<BR>
  <INPUT TYPE="RADIO" NAME="FileAction" VALUE="OVERWRITE" CHECKED>Overwrite
  <INPUT TYPE="RADIO" NAME="FileAction" VALUE="MAKEUNIQUE">Make Unique
  <INPUT TYPE="RADIO" NAME="FileAction" VALUE="SKIP">Skip
```

Listing 33.9 (continued)

```
    <!--- Submit button to submit the form (and upload the file) --->
    <P>
    <INPUT
      TYPE="Submit"
      VALUE="Upload the File">
</FORM>

</BODY>
</HTML>
```

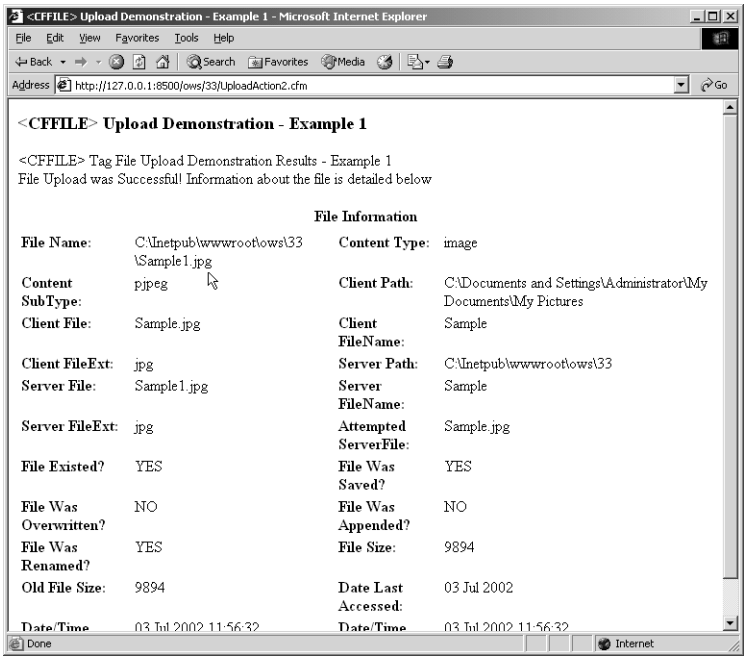
NOTE

The CD-ROM for this book contains the UploadAction2.cfm action page that this form posts its data to. The UploadAction2.cfm file is exactly the same as the original version of the action page (Listing 33.8), except that the `NAMECONFLICT="Overwrite"` attribute has been replaced with `NAMECONFLICT="#FORM.FileAction#"`.

In the form screenshot shown in Figure 33.5, the radio button marked Make Unique was checked. With this option selected, the result of submitting the same file is that the server is forced to create a unique name if the same file is uploaded the second time. The `CFFILE.FileWasRenamed` variable will reflect this with a value of `Yes` (Figure 33.6). In this case, I uploaded a file from my browser that had a filename of Sample.jpg. The second time I uploaded it, the Make Unique behavior kicked in and saved the second version of the file with a filename of Sample1.jpg.

Figure 33.6

Example output with user-specified `NAMECONFLICT` attribute.



The <CFFILE> tag in this example uses data passed from the form in the FileAction field to set the value of the NAMECONFLICT attribute. The field was referenced in the <CFFILE> tag as follows:

```
NAMECONFLICT="#FORM.FILEACTION#"
```

Any of the other attributes can also be set using CFSET variables, FORM attributes, or URL attributes. Note, however, that setting the SOURCE or DESTINATION attribute based on user input can have far-reaching consequences. For security reasons, users should not be permitted to specify SOURCE or DESTINATION attributes using TEXT input fields. The SOURCE and DESTINATION attributes should be set using only template-based code, which is conditionally executed, to provide maximum security.

Manipulating Folders on the Server with <CFDIRECTORY>

Just as <CFFILE> can be used to read, write, and manipulate files, <CFDIRECTORY> can be used to manage directories on the server's drives. Similar to <CFFILE>, <CFDIRECTORY> takes an ACTION attribute, which specifies the action to be performed.

Using <CFDIRECTORY>

To create a directory, the tag is used like this:

```
<CFDIRECTORY
  ACTION="Create"
  DIRECTORY="c:\MyFolders\MyNewFolder">
```

To delete a directory, just modify the ACTION, like this:

```
<CFDIRECTORY
  ACTION="Delete"
  DIRECTORY="c:\MyFolders\MyUnwantedFolder">
```

To rename a directory, use the tag like this:

```
<!--- To rename a directory --->
<CFDIRECTORY
  ACTION="Rename"
  DIRECTORY="c:\MyFolders\MyExistingFolder"
  NEWDIRECTORY="c:\MyFolders\MyNewFolderName">
```

Finally, to get a listing of the contents of a directory (that is, the files and sub-folders that the directory contains), use the tag like this:

```
<CFDIRECTORY
  ACTION="List"
  DIRECTORY="c:\MyFolders\MyExistingFolder"
  SORT="Name ASC"
  FILTER="*.*"
  NAME="MyQueryName">
```

The supported actions for <CFDIRECTORY> are listed formally in Table 33.4, and the tag's attributes are listed in Table 33.5.

Table 33.4 <CFDIRECTORY> Actions

ACTION	DESCRIPTION
CREATE	Creates the directory specified in the DIRECTORY attribute.
DELETE	Deletes the directory specified in the DIRECTORY attribute.
RENAME	Renames the directory specified in the DIRECTORY attribute to the name specified in the NEWDIRECTORY attribute.
LIST	Returns the contents of the directory specified in the DIRECTORY attribute into a query named in the NAME attribute. An optional FILTER can be specified as well, as can a SORT order.

Table 33.5 Additional <CFDIRECTORY> Attributes

ATTRIBUTE	DESCRIPTION
DIRECTORY	Required. Directory on which the action will be taken.
MODE	Optional. Used on Unix versions of ColdFusion to set directory permissions when ACTION="Create". Ignored on Windows. Standard Unix style modes are accepted.
NEWDIRECTORY	Required for ACTION="Rename". Ignored for all other actions. Specifies new name of directory.
NAME	Required for ACTION="List". Ignored for other actions. Specifies name of output query created by the action.
FILTER	Optional. Used with ACTION="List" to filter the files returned in the query. An example is *.txt. Only one filter can be applied at a time. It's ignored for all other actions.
SORT	Optional for ACTION="List". Ignored for other actions. Lists the columns in the query to sort the results with. Specified in a comma-delimited list. Ascending order is the default (ASC). Descending order is specified by the use of DESC. An example is "dirname ASC, name DESC, size".

Getting the Contents of a Directory

When you use <CFDIRECTORY> with ACTION="List", ColdFusion creates a query record set object that contains information about the contents of the directory you specify in the DIRECTORY attribute. The query object is returned to you with the variable name you specify in the NAME attribute. The query object contains one row for every file or subfolder within the directory. The columns of the query object are listed in Table 33.6.

So, for instance, if you provide NAME="FolderContents" in a <CFDIRECTORY> tag, then you can refer to FolderContents.Name to display the name of each file (or subfolder), and FolderContents.Size to refer to its size on your server's drive.

NOTE

On UNIX/Linux servers, there is also a `MODE` column that contains the Octal value that specifies the permissions setting for the directory. For information about octal values, see the UNIX man pages for the chmod shell command.

Table 33.6 Query Columns Populated by `<CFDIRECTORY> ACTION="List"`

COLUMN	DESCRIPTION
`Name`	The name of the file or folder, including the file extension (but not including the full directory path).
`Size`	The size of the file, in bytes.
`Type`	Whether the record represents a file or a folder. If the record represents a file, the value of the `TYPE` column will be File. If it represents a directory, the `TYPE` will be `Dir`.
`DateLastModified`	The date that the file was last modified, as a ColdFusion style date value. You can use this date with `DateFormat()`, `TimeFormat()`, or any of the other date-related functions listed in Appendix C, "ColdFusion Function Reference."
`Attributes`	The file's attributes (read-only, archive, and so on).

Building a Simple File Explorer

Listing 33.10 uses the `ACTION="List"` attribute of `<CFDIRECTORY>` to build a simple web interface for exploring the files and subfolders within the ows folder in your web server's document root. When you visit this page with your web browser, you will see a drop-down list that includes the folders within the ows folder (Figure 33.7). When you select a folder from the list, the page reloads and the files in the selected appear. From there, you can navigate further to any of the selected folder's subfolders, or return to the previous folder using the Parent Folder option in the drop-down list (Figure 33.8).

Figure 33.7

The directory listing provided by `<CFDIRECTORY>` is exposed to the user as a drop-down list.

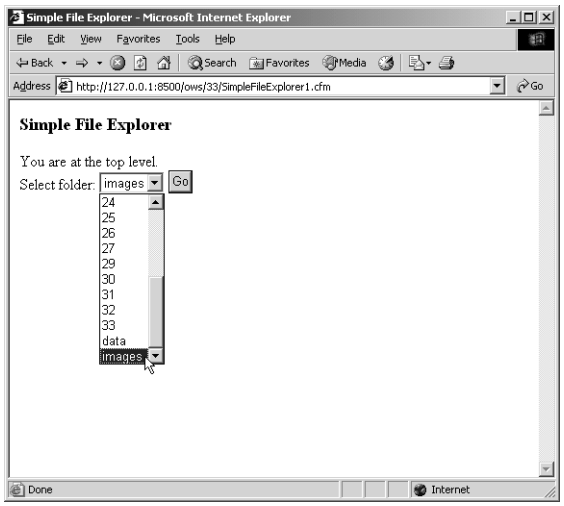
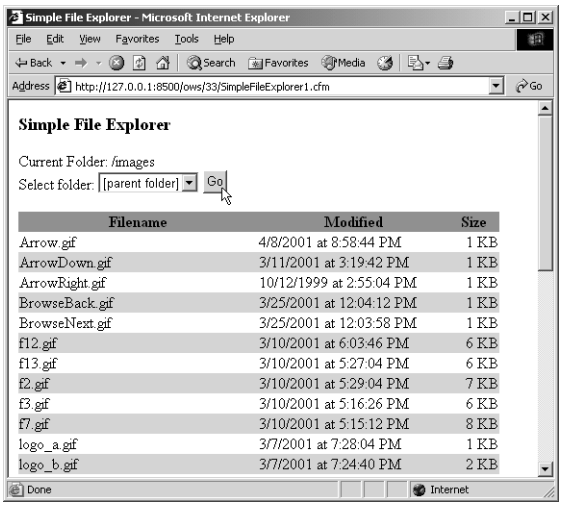


Figure 33.8

Users can view the files in the selected directory, or navigate up and down the folder structure.



NOTE

This example assumes that you are saving the example listings for this book's chapters in the recommended places. That is, the assumption is that there is a folder named ows within your server's Web document root, and that the example listings for Chapter 20 are in ows/20, the listings for Chapter 21 are in ows/21, and so on.

Listing 33.10 `SimpleFileExplorer1.cfm`—Listing Files and Folders Within a Directory

```
<!---
  Filename: SimpleFileExplorer.cfm
    Author:   Nate Weiss (NMW)
    Purpose:  Provides an interface for exploring files and subfolders
              within the ows root
--->


<!--- The user can explore this folder and any nested subfolders --->
<!--- Assume that the parent of the folder that contains this ColdFusion --->
<!--- page (that is, the "ows" folder) should be considered explorable --->
<CFSET BaseFolder = ExpandPath("../")>

<!--- The SubfolderPath variable indicates the currently selected folder --->
<!--- (relative to the BaseFolder). Defaults to an empty string, meaning --->
<!--- that the BaseFolder will be current when the page first appears --->
<CFPARAM NAME="SubfolderPath" TYPE="string" DEFAULT="">

<!--- This variable, then, is the full path of the selected folder --->
<CFSET FolderToDisplay = BaseFolder & SubfolderPath>

<!--- Get a listing of the selected folder --->
<CFDIRECTORY
  DIRECTORY="#FolderToDisplay#"
  NAME="DirectoryQuery"
  SORT="Name ASC"
  FILTER="*.*">
```

Listing 33.10 (CONTINUED)

```
<CFOUTPUT>
  <HTML>
    <HEAD><TITLE>Simple File Explorer</TITLE></HEAD>
  <BODY>
  <H3>Simple File Explorer</H3>

  <!--- Create a simple form for navigating through folders --->
  <FORM ACTION="SimpleFileExplorer1.cfm" METHOD="Post">

    <!--- Show the subfolder path, unless already at top level --->
    <CFIF SubfolderPath EQ "">
      You are at the top level.<BR>
    <CFELSE>
      Current Folder: #SubfolderPath#<BR>
    </CFIF>

    <!--- Provide a drop-down list of subfolder names --->
    Select folder:
    <SELECT NAME="SubfolderPath" onchange="this.form.submit()">

      <!--- Provide an option to go up one level to the parent folder, --->
      <!--- unless already at the BaseFolder --->
      <CFIF ListLen(SubfolderPath, '/') GT 0>
        <CFSET ParentFolder = ListDeleteAt(SubfolderPath, ListLen(SubfolderPath,
        ➥'/'), '/')>
        <OPTION VALUE="#ParentFolder#">[parent folder]
      </CFIF>

      <!--- For each record in the query returned by <CFDIRECTORY> --->
      <CFLOOP QUERY="DirectoryQuery">
        <!--- If the record represents a subfolder, list it as an option --->
        <CFIF Type EQ "Dir">
          <OPTION VALUE="#SubfolderPath#/#Name#">#Name#
        </CFIF>
      </CFLOOP>
    </SELECT>

    <!--- Submit button to navigate to the selected folder --->
    <INPUT TYPE="Submit" VALUE="Go">
  </FORM>


  <!--- Use Query of Queries (In Memory Query) to get a subset of --->
  <!--- the query returned by <CFDIRECTORY>. This new query object --->
  <!--- will hold only the file records, not any subfolder records --->
  <CFQUERY DBTYPE="query" NAME="FilesQuery">
    SELECT * FROM DirectoryQuery
    WHERE TYPE = 'File'
  </CFQUERY>


  <!--- If there is at least one file to display... --->
  <CFIF FilesQuery.RecordCount GT 0>
    <!--- Display the files in a simple HTML table --->
    <TABLE WIDTH="500" BORDER="0" CELLPADDING="1" CELLSPACING="0">
      <TR BGCOLOR="CornflowerBlue">
```

Listing 33.10 (continued)

```
            <TH>Filename</TH>
            <TH>Modified</TH>
            <TH>Size</TH>
          </TR>

          <!--- For each file... --->
          <CFLOOP QUERY="FilesQuery">
            <!--- Use alternating colors for the table rows --->
            <!--- This is explained in the "Next N" examples from Chapter 21 --->
            <CFIF FilesQuery.CurrentRow MOD 2 EQ 0>
              <CFSET RowColor = "LightGrey">
            <CFELSE>
              <CFSET RowColor = "White">
            </CFIF>

            <!--- Display the file details --->
            <TR BGCOLOR="#RowColor#">
              <!--- File name --->
              <TD WIDTH="250">
                #Name#
              </TD>
              <!--- File modification date and time --->
              <TD WIDTH="200">
                #DateFormat(DateLastModified, "m/d/yyyy")#
                at
                #TimeFormat(DateLastModified, "h:mm:ss tt")#
              </TD>
              <!--- File size --->
              <TD WIDTH="50" ALIGN="right">
                #Ceiling(Size / 1024)# KB
              </TD>
            </TR>
          </CFLOOP>
        </TABLE>
      </CFIF>

      </BODY>
      </HTML>
    </CFOUTPUT>
```

First, the `ExpandPath()` function is used to create a variable called `BaseFolder` that holds the path to the ows folder on your server's drive. The actual value of this variable when your page executes will likely be `c:\CFusionMX\wwwroot\ows`, `c:\inetpub\wwwroot\ows`, or something similar, depending on the web server you are using. For common sense security reasons, the user will only be able to explore files and directories within the `BaseFolder`. If you want the user to be able to explore some other folder, perhaps outside of your server's document root, you can just hardcode the `BaseFolder` variable with the location of that folder.

Next, the `<CFPARAM>` tag is used to declare a variable named `SubfolderPath` and give it a default value of an empty string. The idea is that this variable will indicate which subfolder within the `BaseFolder` that the user wants to explore. If a URL or FORM parameter called `SubfolderPath` is provided to the page, that value will be used; otherwise it is assumed that the page is appearing for the first time. If the user has selected the subfolder named images, then the value of `SubfolderPath` will be `/20`.

The FolderToDisplay variable is then created by concatenating the BaseFolder together with the SubfolderPath. This variable, then, holds the full path to the folder the user wants to explore; this is what will be supplied to the <CFDIRECTORY> tag to obtain the contents of the folder. So, if the user If the user has selected the subfolder named images, then the value of FolderToDisplay will be c:\CFusionMX\wwwroot\ows\images, c:\inetpub\wwwroot\ows\images, or something similar, depending on what web server you are using.

Now the <CFDIRECTORY> tag can be used to get a listing of all the files and subfolders within the selected folder. This will result in a query object called DirectoryQuery, which will contain the columns listed in Table 33.6.

Near the middle of this listing, a <CFLOOP> tag is used to loop over the DirectoryQuery query, generating an <OPTION> tag for each subfolder within the current folder (as shown in Figure 33.7). Within the loop, a <CFIF> test is used to only output options for rows where the Type column is set to Dir. This step is necessary because the query object may contain rows for subfolders and other rows for individual files. The <CFIF> test effectively filters the query object so that only rows that represent folders are processed.

Another way to filter a directory query object by type (that is, to only include files or folders) is to use ColdFusion MX's Query of Queries feature (also called In Memory Query) that you learned about in Chapter 29, "More About SQL and Queries". This strategy is used in the second half of this listing, to create a filtered version of DirectoryQuery (called FilesQuery), that only contains records for files, not subfolders. Once that's done, outputting the actual information about files is a simple matter. A <CFLOOP> block is used to output the file information in a simple HTML table, displaying the values of each record's Name, Size, and DateLastModified columns. Note that the ordinary DateFormat(), TimeFormat(), and Ceiling() functions are used to display the data attractively.

This book's CD-ROM also contains a SimpleFileExplorer2.cfm page that adds the ability for the user to add and remove directories to the server's drive. This second version of the page is only slightly more complex than the one shown in Listing 33.10. As a learning exercise, you are encouraged to take a look at the listing and study how it works.

Building an Application to Manage Files

Now that you have seen the <CFFILE> tag used in simple examples, you can modify the sample application that combines file upload capabilities, directory listings, and file manipulation. The example application for this section will be a Web based interface for updating information in the Actors table within the Orange Whip Studios database.

Three CFML templates will be created for this example application. The first template provides a list of actors from which to choose. The second template is a simple form that allows you to edit actor information or upload a photo of the actor. The third template accepts the uploaded photos and makes any appropriate changes to the database.

Let's start by creating the template to provide the actor list, which is shown in Listing 33.11.

Listing 33.11 `ActorPhotoList.cfm`—Actor Listing Template

```
<!---
  Filename:  ActorPhoto1.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Creates an interface for uploading actor photos
--->


<!--- Query the database to get a listing of actors to choose from --->
<CFQUERY DATASOURCE="ows" NAME="ActorsQuery">
  SELECT ActorID, NameFirst, NameLast
  FROM Actors
  ORDER BY NameLast, NameFirst
</CFQUERY>

<!--- Location of the directory where actor photos are stored --->
<!--- The folder name is ActorPhotos, within this page's folder --->
<CFSET ActorPhotoDir = ExpandPath("ActorPhotos")>

<!--- If the photo directory does not exist yet... --->
<CFIF NOT DirectoryExists(ActorPhotoDir)>
  <!--- ...go ahead and create the photo directory --->
  <CFDIRECTORY
    ACTION="Create"
    DIRECTORY="#ActorPhotoDir#">
</CFIF>



<HTML>
<HEAD>
  <TITLE>Maintain Actor Photos</TITLE>
</HEAD>
<BODY>
<H2>Maintain Actor Photos</H2>
<H3>Click on Actor Name to Add Photo</H3>

<TABLE BORDER="1" WIDTH="300">
  <TR>
    <TH>Name</TH>
    <TH>Photo</TH>
  </TR>

  <!--- For each actor in the database... --->
  <CFOUTPUT QUERY="ActorsQuery">

    <!--- Get a listing of any current image files for this actor --->
    <CFDIRECTORY
      ACTION="List"
      DIRECTORY="#ActorPhotoDir#"
      NAME="ExistingPhotos"
      FILTER="Actor#ActorsQuery.ActorID#Photo.*">

    <TR>
      <TD>
```

Listing 33.11 (CONTINUED)

```
        <!--- Provide link to upload page, passing actor's ID in the URL --->
        <A HREF="ActorPhotoForm.cfm?ActorID=#ActorID#">#NameFirst# #NameLast#</A>
      </TD>
      <TD HEIGHT="50">
        <!--- If there is a photo for this actor --->
        <CFIF ExistingPhotos.RecordCount NEQ 0>
          <!--- Display a thumbnail view of the photo --->
          <A HREF="ActorPhotoForm.cfm?ActorID=#ActorID#">
          <IMG
            SRC="ActorPhotos/#ExistingPhotos.Name#"
            WIDTH="50"
            HEIGHT="50"
            BORDER="0"
            ALT="Click to edit"></A>
        <CFELSE>
        [no photo]
        </CFIF>
      </TD>
    </TR>
  </CFOUTPUT>
</TABLE>

</BODY>
</HTML>
```

The template created in Listing 33.11 selects a list of actors from the `Actors` table. A link is created for each actor in the `Actors` table; the link takes you to another template, where the actor's information can be updated or a new photo can be provided.

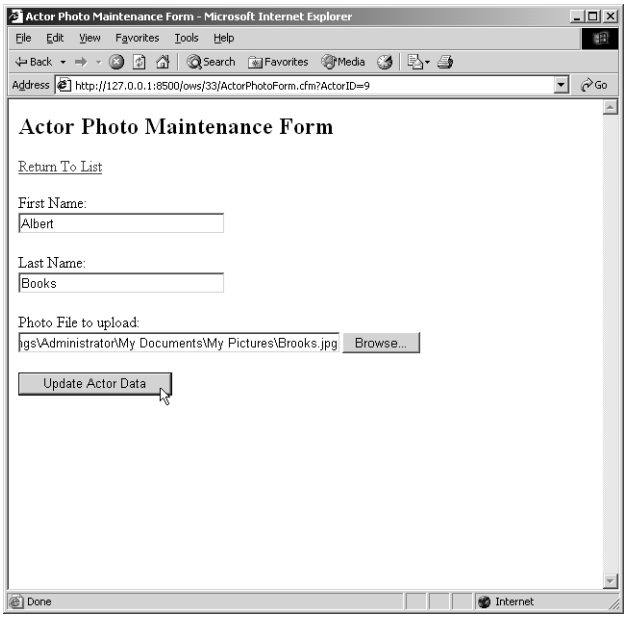
In most respects, this is just an ordinary data-display page. The only twists involve the <CFDIRECTORY> tag, which is used in two different places to enable the display of thumbnail-sized photographs next to each actor's name. At the top of the listing, a variable called ActorPhotoDir is created, which contains the path to the folder where the actor photos will be stored. The folder is called ActorPhotos, located within the folder that contains this listing. If the folder does not exist already, it is created with the <CFDIRECTORY> tag.

Then, within the loop that displays each actor's name, the <CFDIRECTORY> tag is used again to find out if there are any photos for the actor currently being processed. Actor photos will be stored with filenames based on the actor's ID number. Users will be upload photos of any type (GIF, JPEG, PNG, and so on), so the FILTER attribute is used with a * wildcard for the extension. For actor number 5, for instance, the ActorPhotos folder might contain a file called Actor5photo.gif; for actor number 12, there might be a Actor12Photo.jpg or Actor12Photo.png file instead.

In any case, the <CFDIRECTORY> tag will find whatever relevant files are present (if any). If the resulting query's RecordCount property is greater than zero, that means that there is a photo for the actor. If so, the photo is displayed next to the actor's name as a thumbnail (displayed at a size of 50 pixels wide by 50 pixels high).

Listing 33.12 shows the ColdFusion page that the user lands at when they click on an actor's name. The form created in this listing code serves two purposes. First, an actor's photo can be updated via file uploading. Second, the actor's record in the database can be updated, just like an ordinary data update form. In other words, the internal details about how the information is stored (the fact that the photo is stored as a discrete file, while the actor's name is stored in the database) is hidden from the user. The distinction doesn't matter to the user, so there is no reason not to provide a single form to manage everything about an actor's information (Figure 33.9).

Figure 33.9
Actor photo upload form.



Listing 33.12 `ActorPhotoForm.cfm`—Actor Photo Maintenance Form

```
<!---
  Filename:  ActorPhoto1.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Creates an interface for uploading actor photos
--->

<!--- Insist that an ActorID parameter is passed in the URL --->
<CFPARAM NAME="URL.ActorID" TYPE="numeric">

<!--- Query the database for information about the specified actor --->
<CFQUERY NAME="ActorQuery" DATASOURCE="ows">
  SELECT ActorID, NameLast, NameFirst
  FROM Actors
  WHERE ActorID = #Val(URL.ActorID)#
</CFQUERY>

<!--- Location of the directory where actor photos are stored --->
<!--- The folder name is ActorPhotos, within this page's folder --->
<CFSET ActorPhotoDir = ExpandPath("ActorPhotos")>

<!--- Get a listing of any current image files for this actor --->
<CFDIRECTORY
  ACTION="List"
  DIRECTORY="#ActorPhotoDir#"
  NAME="ExistingPhotos"
  FILTER="Actor#URL.ActorID#Photo.*">

<HTML>
<TITLE>Actor Photo Maintenance Form</TITLE>
<BODY>
<H2>Actor Photo Maintenance Form</H2>

<!--- Link to return to the list of actors --->
<A HREF="ActorPhotoList.cfm">Return To List</A>

<CFOUTPUT>

  <!--- This form submits to ActorPhotoAction.cfm, --->
  <!--- passing the ActorID along in the URL. --->
  <CFFORM
    ACTION="ActorPhotoAction.cfm?ActorID=#URL.ActorID#"
    ENCTYPE="multipart/form-data"
    METHOD="Post">

    <!--- Text input field for first name --->
    <P>First Name:<BR>
    <CFINPUT
      TYPE="Text"
      NAME="NameFirst"
      VALUE="#ActorQuery.NameFirst#"
      SIZE="30"
      MAXLENGTH="50"
      REQUIRED="Yes"
      MESSAGE="Please don't leave the first name blank.">

    <!--- Text input field for last name --->
    <P>Last Name:<BR>
    <CFINPUT
```

Listing 33.12 (continued)

```
      TYPE="Text"
      NAME="NameLast"
      VALUE="#ActorQuery.NameLast#"
      SIZE="30"
      MAXLENGTH="50"
      REQUIRED="Yes"
      MESSAGE="Please don't leave the last name blank.">

    <!--- File field for user to select or specify a filename --->
    <P>Photo File to upload:<BR>
    <INPUT
      NAME="PhotoFile"
      SIZE="50"
      TYPE="FILE"><BR>

    <!--- If there are any existing photos for this user, --->
    <!--- provide a checkbox for deleting the existing photo --->
    <CFIF ExistingPhotos.RecordCount GT 0>
      <INPUT
        TYPE="Checkbox"
        NAME="DeletePhoto">Delete existing photo
    </CFIF>

    <!--- Submit button to submit the form (and upload the file) --->
    <P>
    <INPUT
      TYPE="SUBMIT"
      VALUE="Update Actor Data">


    <!--- Display the photo if available --->
    <CFIF ExistingPhotos.RecordCount GT 0>
      <P>Existing photo shown below:<BR>
      <IMG
        SRC="ActorPhotos/#ExistingPhotos.Name#"
        BORDER="0"
        ALT="Photo of #ActorQuery.NameFirst# #ActorQuery.NameLast#">
    </CFIF>

  </CFFORM>
</CFOUTPUT>

</BODY>
</HTML>
```

This template is a simple modification of the first file upload template you wrote. First, the `URL.ActorId` variable is used to retrieve the actor's name from the `Actors` table. The actor's first and last names are displayed in simple text input fields using the `<CFINPUT>` tag. A `TYPE="File"` field is also provided, where the user can select a photo of the actor. When the form is submitted, the photo and the first/last name information is all sent to the server, where it will be processed by the template in Listing 33.13.

This listing also uses the `<CFDIRECTORY>` tag to find out if there are any existing photos of the actor, using the same basic logic that was used in Listing 33.11. If there is a photo, a checkbox is added to the form to allow the user to delete the existing photo (the photo is also displayed at full size under the form).

Listing 33.13 is the action page that is executed when the user submits the form shown in Figure 33.9.

Listing 33.13 ActorPhotoAction.cfm—Actor Photo Upload Process Template

```
<!---
  Filename:  ActorPhoto1.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Creates an interface for uploading actor photos
--->

<!--- Insist that an ActorID parameter is passed in the URL --->
<CFPARAM NAME="URL.ActorID" TYPE="numeric">
<!--- Also, make sure the expected form fields are present --->
<CFPARAM NAME="FORM.NameFirst" TYPE="string">
<CFPARAM NAME="FORM.NameLast" TYPE="string">
<CFPARAM NAME="FORM.PhotoFile" TYPE="string">

<!--- Location of the directory where actor photos are stored --->
<!--- The folder name is ActorPhotos, within this page's folder --->
<CFSET ActorPhotoDir = ExpandPath("ActorPhotos")>



<!--- Function for deleting the existing photos of the actor --->
<CFFUNCTION NAME="DeleteExistingPhotos">
  <CFARGUMENT NAME="ActorID" TYPE="numeric" REQUIRED="Yes">

  <!--- Get a listing of any current image files for this actor --->
  <CFDIRECTORY
    ACTION="List"
    DIRECTORY="#ActorPhotoDir#"
    NAME="ExistingPhotos"
    FILTER="Actor#ARGUMENTS.ActorID#Photo.*">

  <!--- For each existing file, if any... --->
  <CFLOOP QUERY="ExistingPhotos">
    <!--- Delete the existing file --->
    <CFFILE
      ACTION="Delete"
      FILE="#ActorPhotoDir#\#ExistingPhotos.Name#">
  </CFLOOP>
</CFFUNCTION>



<!--- Query the database for information about the specified actor --->
<CFQUERY NAME="ActorQuery" DATASOURCE="ows">
  SELECT ActorID, NameLast, NameFirst
  FROM Actors
  WHERE ActorID = #Val(URL.ActorID)#
</CFQUERY>


<!--- If the actor's first or last names have been edited... --->
<CFIF (ActorQuery.NameFirst NEQ FORM.NameFirst)
   OR (ActorQuery.NameLast  NEQ FORM.NameLast)>

   <!--- Update the actor's record in the database --->
   <CFQUERY DATASOURCE="ows">
```

Listing 33.13 (continued)

```
        UPDATE Actors
        SET
          NameFirst = '#FORM.NameFirst#',
          NameLast  = '#FORM.NameLast#'
        WHERE ActorID = #URL.ActorID#
    </CFQUERY>
</CFIF>



<!--- If the user has specified a file to upload --->
<CFIF FORM.PhotoFile NEQ "">

  <!--- Make sure only one person is working with this actor's photos --->
  <CFLOCK
    NAME="ActorPhotoUploads#URL.ActorID#"
    TYPE="Exclusive"
    TIMEOUT="10">

    <!--- Accept the file upload --->
    <CFFILE
      DESTINATION="#GetTempDirectory()#"
      ACTION="UPLOAD"
      NAMECONFLICT="MakeUnique"
      FILEFIELD="PhotoFile"
      ACCEPT="image/*">

    <!--- Delete the existing photographs --->
    <CFSET DeleteExistingPhotos(URL.ActorID)>

    <!--- Create a new name for the uploaded file, based on the actor's ID --->
    <CFSET NewFile = "Actor#URL.ActorID#Photo.#File.ServerFileExt#">

    <!--- Go ahead and rename the uploaded file --->
    <CFFILE
      ACTION="RENAME"
      SOURCE="#CFFILE.ServerDirectory#/#CFFILE.ServerFile#"
      DESTINATION="#ActorPhotoDir#/#NewFile#">

  </CFLOCK>

<CFELSEIF IsDefined("FORM.DeletePhoto")>
  <!--- Make sure only one person is working with this actor's photos --->
  <CFLOCK
    NAME="ActorPhotoUploads#URL.ActorID#"
    TYPE="Exclusive"
    TIMEOUT="10">

    <!--- Delete the existing photographs --->
    <CFSET DeleteExistingPhotos(URL.ActorID)>
  </CFLOCK>

</CFIF>


<!--- Now that the edit is complete, send the user back to the form page --->
<CFLOCATION
  URL="ActorPhotoForm.cfm?ActorID=#URL.ActorID#">
```

At the top of this listing, a user defined function called `DeleteExistingPhotos()` is created. When called, this function will delete all photos from the ActorPhotos folder that correspond to the ActorID argument provided. This is accomplished using a simple `<CFDIRECTORY>` tag (which is used nearly identically as in the two previous listings) to get a listing of any corresponding actor photos, then looping through the photo records, deleting each photo with the `<CFILE>` tag.

Next, a query is executed to select the first and last name of the selected actor from the database. If the user has edited the actor's name (that is, if the values from the form submission no longer match the values in the database), the Actors table is updated with a simple `UPDATE` query.

Next, the listing checks the value of the `FORM.PhotoFile` field. If it is not empty, then the user is submitting a file to be uploaded. The upload is accepted using the `<CFFILE>` tag with `ACTION="Upload"`. Please note that the `GetTempDirectory()` function is used to so that the file is uploaded into the server's temporary directory, rather than the ActorPhotos folder. The `DeleteExistingPhotos()` function is then called to delete any existing photos of the actor. Finally, the uploaded file is moved from the temporary folder to the ActorPhotos folder, where it belongs. As it is moved, the file is also given an appropriate filename (based on the actor's ID number, but retaining the file's original extension as reported by the `CFFILE.ServerFileExt` variable).

NOTE

The `<CFFILE>` tag's `ACCEPT` attribute is set to allow only images files to be uploaded to the server. Because the subtype of the `ACCEPT` attribute is specified with the * wildcard, any type of image can be uploaded (GIF, JPEG, PNG, and so on).

If a file is not being uploaded, this listing checks to see if the Delete Existing Photo checkbox was checked on the form by testing for the presence of the `FORM.DeletePhoto` variable. If so, the `DeleteExistingPhotos()` function is called to delete any existing photos of the actor.

Protecting File Accesses with <CFLOCK>

It's worth noting that both of the file-manipulation blocks in Listing 33.13 are protected with the <CFLOCK> tag. The idea is that no two users should be able to alter an actor's photo at the same time, simply to avoid any potential problems that might arise if one user's page request is trying to delete a photo while another user's request is trying to update the photo via a new upload.

The NAME attribute of the <CFLOCK> tag is set to a combination of the string ActorPhotoUploads and the actor's actual ID number. This will have the effect of allowing two users to alter two different actors' photos at the same time, but will not allow two users to alter the same actor's photo at the same time.

I recommend that you use <CFLOCK> in a similar fashion whenever your code is allowing users to edit files or directories in such a way that simultaneous page accesses could be a problem. In general, you will always want to use the NAME attribute of the <CFLOCK> tag for file-related locking, not the SCOPE attribute.

For more information about <CFLOCK>, please consult Appendix B, ColdFusion Tag Reference, or the discussion on locking in Chapter 16, Introducing the Web Application Framework.

Executing Programs on the Server with <CFEXECUTE>

ColdFusion provides a powerful tool for interacting with the operating system in the <CFEXECUTE> tag. It enables the execution of server processes at the command-line level. It is powerful yet simple.

NOTE

Executing processes on the server can have disastrous consequences, so extreme care should be taken to control access to templates that use the <CFEXECUTE> tag. Arbitrary user input of arguments to the tag should be prohibited.

Listing 33.14 shows the basic arguments for the <CFEXECUTE> tag.

Listing 33.14 <CFEXECUTE> Arguments

```
<CFEXECUTE
  NAME="Application name"
  ARGUMENTS="Command line arguments"
  OUTPUTFILE="Output file name"
  TIMEOUT="Timeout interval in seconds" >
```

Table 33.7 shows the definitions of the arguments and attributes for the <CFEXECUTE> tag.

NOTE

On Windows systems, the NAME argument must contain the fully qualified path to the program to be executed, including the extension (e.g.: C:\WINNT\SYSTEM32\IPCONFIG.EXE).

Several things are worth noting about the attributes of the <CFEXECUTE> tag.

Table 33.7 <CFEXECUTE> Tag Syntax

ATTRIBUTE	DESCRIPTION
NAME	Required. The fully qualified name of the application to execute.
ARGUMENTS	Optional. Command-line arguments to be passed to the program.
OUTPUTFILE	Optional. File in which output of program will be written. If you don't provide this attribute, the output of the program will be simply be included in the current ColdFusion page.
TIMEOUT	Optional. Indicates how long in seconds ColdFusion will wait for the process to complete. Values must be integers greater than or equal to 0.

If ARGUMENTS is passed as a string, it is processed in the following ways:

- On Windows systems, the entire string is passed to the Windows process for parsing.
- On Unix, the string is tokenized into an array of arguments. The default token separator is a space; arguments with embedded spaces can be delimited by double quotes.

If ARGUMENTS is passed as an array, it is processed as follows:

- On Windows systems, the array elements is concatenated into a string of tokens, separated by spaces. This string is then passed to the Windows process.
- On Unix, the elements of the ARGUMENTS array is copied into a corresponding array of `exec()` arguments.

If TIMEOUT and OUTPUTFILE are not provided as attributes to the tag, the resulting output from the executed process is ignored.

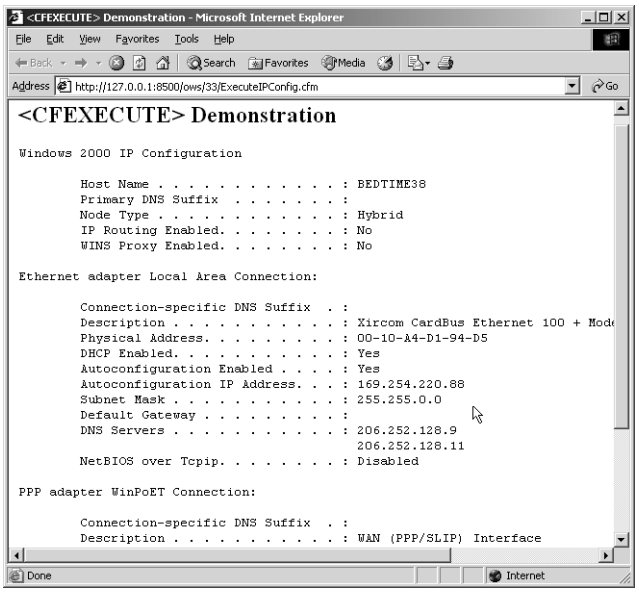
The TIMEOUT attribute is used to determine whether ColdFusion should execute the called process asynchronously (that is, spawn process and continue) or synchronously (spawn process and wait). A value of `0` spawns the process asynchronously, with the ColdFusion execution picking up at the next line of CFML code immediately. Any positive integer value causes the process to be spawned synchronously, with ColdFusion waiting for TIMEOUT seconds before proceeding.

If errors occur during the process, exceptions are thrown that can be handled with <CFTRY> and <CFCATCH> (as discussed in Chapter 31, Error Handling). These exceptions are:

- If the application name is not found, an `Application File Not Found` exception is thrown.
- If the output file cannot be opened, an `Output File Cannot Be Opened` exception is thrown.
- If ColdFusion does not have permissions to execute the process, a security exception is thrown.

Listing 33.15 shows an example of using the <CFEXECUTE> tag to determine IP configuration information about the server using the IPCONFIG utility (Figure 33.10).

Figure 33.10
Output from the <CFEXECUTE> example.



Listing 33.15 `ExecuteIPConfig.cfm`—`<CFEXECUTE>` Example Showing Output from `IPCONFIG`

```
<!---
  Filename:  ExecuteIPConfig.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Demonstrates use of the Windows IPCONFIG utility
--->


<HTML>
<HEAD>
   <TITLE>&lt;CFEXECUTE&gt; Demonstration</TITLE>
</HEAD>

<BODY>
<H2>&lt;CFEXECUTE&gt; Demonstration</H2>


<!--- Set up output file --->
<CFSET OutFile = GetTempFile(GetTempDirectory(), "ipc")>

<!--- Call the system utility, with output placed in the file --->
<CFEXECUTE
  NAME="c:\winnt\system32\ipconfig.exe"
  ARGUMENTS="/ALL"
  TIMEOUT="15"
  OUTPUTFILE="#OutFile#"></CFEXECUTE>
```

Listing 33.15 (CONTINUED)

```
<!--- Read the file for display --->
<CFFILE
  ACTION="READ"
  FILE="#OutFile#"
  VARIABLE="FileContent">

<!--- Delete the file, since it is no longer needed --->
<CFFILE
  ACTION="DELETE"
  FILE="#OutFile#">

<!--- Display the contents of the file --->
<CFOUTPUT>
  #HTMLCodeFormat(FileContent)#
</CFOUTPUT>

</BODY>
</HTML>
```

The code in Listing 33.15 is fairly simple and straightforward. Using techniques learned earlier in the chapter, `GetTempFile()` is used to get a filename to be passed as the `OUTPUTFILE` argument to `<CFEXECUTE>`. The `NAME` attribute is set to the fully qualified pathname of the executable that is to be run. In this case, that's the `IPCONFIG` utility, which is traditionally located at `c:\winnt\system32\ipconfig.exe` on Windows systems).

The `ARGUMENTS` attribute is set to `"/ALL"`, which tells the `IPCONFIG` utility to return information about all defined interfaces. Lastly, the `TIMEOUT` attribute is set to `15` seconds, indicating that the process should be spawned in a synchronous fashion. The output from the process is then read into a variable using `<CFFILE>`. The temporary file is then deleted, since it is no longer needed.

NOTE

Of course, if you are using a non-Windows system, this call to IPCONFIG won't work. The `<CFEXECUTE>` tag, by its very nature, leads to application code that will probably not work across different operating systems.

NOTE

It is also possible to code this page such that the temporary file is not needed at all, by simply omitting the `OUTPUTFILE` attribute. The ExecuteIPConfig2.cfm page (on the CD-ROM) is a revised version of Listing 33.15 that doesn't use a temporary file.

`<CFEXECUTE>` provides a powerful set of functionality, but its use should be carefully evaluated because any server process has the potential to affect the stability of the server. There are many potential uses for `CFEXECUTE`, including the capability to

- Submit batch processes to legacy command-line applications
- Use CF to communicate with external processes via the command line
- Execute CF templates asynchronously using batch files and the `CFML.exe` stub file

Interacting with the System Registry Using <CFREGISTRY>

If you are using a Windows server for ColdFusion, you may be interested in another powerful means of interacting with the operating system: the <CFREGISTRY> tag. As its name implies, <CFREGISTRY> provides access to the Windows Registry.

NOTE

<CFREGISTRY> has the potential for causing serious harm to the stability of a server, if improperly used. In fact, it can be disabled in the ColdFusion Administrator as a precaution. Bottom Line: Use extreme care when using the <CFREGISTRY> tag to modify the system Registry.

What Is the Registry?

The Registry is a system database that primarily holds information about where things are located in the operating system. Programs, paths, default values, and more are stored within it. Because it is a system database, it has very fast access to information. The only problem is that it is not designed for real database work. This is one of the reasons client variables can better be stored in a database rather than in the Registry, which is the default setting.

NOTE

Even though the Registry is a Windows-only construct, a simulation is provided on Unix platforms. This simulation has the same effect as the Windows Registry, and the tag reacts the same way to both.

NOTE

If you are not familiar with the Windows Registry, you should not change any of the settings. This cannot be recommended strongly enough! If you decide to explore the Windows Registry, be sure to have a good Registry book on hand, such as Troubleshooting and Configuring the Windows NT/95 Registry (Sams Publishing, ISBN: 0-672-31066-X).

One important note is that the Registry is a system database for Windows machines. The structure and job of the Registry exist only for the Windows NT and Windows NT/95/98 versions of ColdFusion. Therefore, the <CFREGISTRY> tag has very limited use with non-Windows servers. Certain terms are used in conjunction with the Windows Registry that you need to become familiar with to use the <CFREGISTRY> tag:

- **Key**—This is the same as a directory in a filesystem. It holds subkeys (subdirectories) and entries (files).
- **Entry**—A variable within a key that holds a data value. Entries do not change unless they are deleted and re-created. Only their content can be altered.
- **Value**—The data contained within an entry. ColdFusion does not allow binary data to be set or retrieved from values.
- **Branch**—A specific path mapping from the root of a Registry tree to a specific subkey.

NOTE

Previous versions of ColdFusion used the system Registry to store information about how ColdFusion was configured. Changing settings in the ColdFusion Administrator was editing the Registry behind the scenes. In ColdFusion MX, all server settings are now stored in XML files and other disk-based means (for details about these files, see Appendix F, ColdFusion MX Directory Structure). Therefore, while it was common to use <CFREGISTRY> to find the current values of certain ColdFusion settings programmatically with older versions of ColdFusion, that practice will no longer have the same effect.

Like the <CFFILE> and <CFDIRECTORY> tags that you've already learned about in this chapter, the <CFREGISTRY> tag supports an ACTION attribute that enables you to perform various Registry-related tasks. The possible ACTION values are listed in Table 33.8, and are discussed in the following sections.

Table 33.8 <CFREGISTRY> Actions

ATTRIBUTE	DESCRIPTION
ACTION="Get"	Reads a particular value from the Registry.
ACTION="GetAll"	Gets a listing of available values from a particular branch of the Registry.
ACTION="Set"	Changes or creates a value in the Registry.
ACTION="Delete"	Removes a value from the Registry.

Get Action

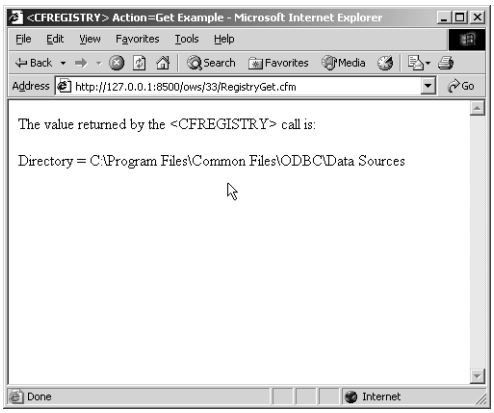
The Get action of <CFREGISTRY> is similar to the <CFSET> tag. It sets a variable with a value, derived from the Registry. Table 33.9 shows the attributes for the <CFREGISTRY> tag when the action is set to GET.

Table 33.9 <CFREGISTRY> Tag Attributes with GET Action

ATTRIBUTE	DESCRIPTION
ENTRY	Required. The Registry value to be accessed
TYPE	Optional. The type of data you want to access. TYPE="String" (the default) returns a string value (default). TYPE="DWord" returns a numeric value. TYPE="Key" returns the key's value using its default type.
VARIABLE	Required. Variable into which <CFREGISTRY> places the value.

One important note is that this tag tries to set the variable with the entry value no matter what. So, if the entry value does not exist, it fails. However, the GetAll section of this chapter shows a way around this. Listing 33.16 uses <CFREGISTRY> with the Get action to retrieve the company name to whom the ColdFusion server is registered (Figure 33.11).

Figure 33.11
Output from the <CFREGISTRY> ACTION=Get example.



Listing 33.16 RegistryGet.cfm—<CFREGISTRY> with the Get Attribute

```
<!---
  Filename:  RegistryGet.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Demonstrates use of the <CFREGISTRY> tag
--->

<html>
<head>
   <title>&lt;CFREGISTRY&gt; Action=Get Example</title>
</head>

<body>

<!---
This example uses CFREGISTRY with the Get Action to retrieve the default
directory for File ODBC Datasources
--->
<CFREGISTRY
  ACTION="GET"
  BRANCH="HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC File DSN"
  ENTRY="DefaultDSNDir"
  VARIABLE="ODBC_Information"
  TYPE="String">

The value returned by the &lt;CFREGISTRY&gt; call is:
<p>
<cfoutput>
  Directory = #ODBC_Information#<br><br>
</cfoutput>

</body>
</html>
```

The GetAll Action

The GetAll action is the broadest of the <CFREGISTRY> actions. It searches and returns all values within a key. This capability covers both Key and Entry values. The result of this action is a standard ColdFusion query with the name specified by the NAME attribute. Table 33.10 shows the <CFREGISTRY> attributes to use with the GetAll action.

Table 33.10 Attributes for the GetAll Action of the <CFREGISTRY> Tag

ATTRIBUTE	DESCRIPTIONS
BRANCH	Required. The branch of the registry that you want to retrieve information from.
TYPE	Optional. The type of data you want to access. TYPE="String" returns all string values within the branch of the registry. TYPE="DWord" returns all numeric values within the branch. TYPE="Key" returns all nested sub-keys within the branch. TYPE="Any" returns all of the above.
NAME	Required. The name of the resultset to contain returned keys and values.
SORT	Optional. Used to sort query column data returned with ACTION="GETALL". Ignored for all other actions. Sorts on Entry, Type, and Value fields as text. Any combination of columns from a query output can be specified in a comma-separated list. ASC (ascending) or DESC (descending) can be specified as qualifiers for column names. ASC is the default.

When run, this version of the <CFREGISTRY> tag returns a query with three columns:

- **Entry**—The name of the key or entry.
- **Type**—The data type (refer to Table 33.10).
- **Value**—If the type is not a key, this holds the value of the entry.

Listing 33.17 shows an example of using the GetAll action to read the registration information from the Registry and display it (Figure 33.12).

Listing 33.17 RegistryGetAll.cfm—<CFREGISTRY> Action=GetAll Example

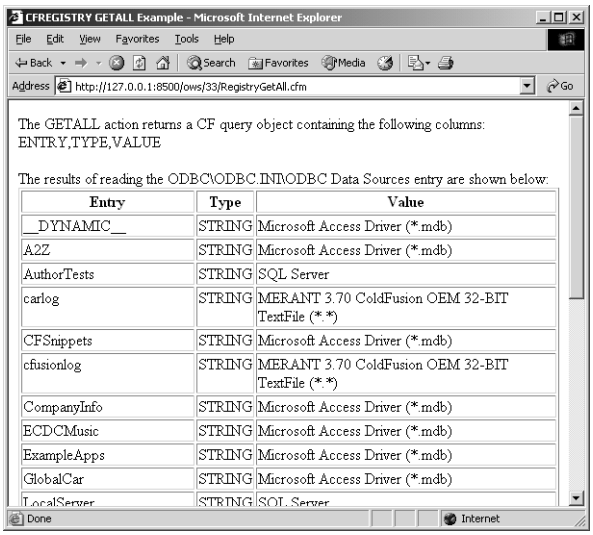
```
<!---
  Filename:  RegistryGetAll.cfm
  Edited By: Nate Weiss (NMW)
  Purpose:   Demonstrates use of the <CFREGISTRY> tag
--->


<html>
<head>
    <title>CFREGISTRY GETALL Example</title>
</head>

<body>

<!--- Retreive information from the Windows Registry --->
```

Listing 33.17 (continued)

```
<CFREGISTRY
  ACTION="GETALL"
  BRANCH="HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources"
  NAME="ODBC_Information"
  TYPE="Any"
  SORT="entry ASC, value DESC">

<!--- Display message about information returned by <CFREGISTRY> --->
The GETALL action returns a CF query object
containing the following columns:<BR>
<cfoutput>#ODBC_Information.columnlist#</cfoutput>

<p>
The results of reading the ODBC\ODBC.INI\ODBC Data Sources
entry are shown below:
<table border>
  <tr>
    <th valign="left">Entry</th>
    <th valign="left">Type</th>
    <th valign="left">Value</th>
  </tr>

  <!--- For each value in the specified branch of the Registry... --->
  <cfoutput query="ODBC_Information">
    <tr>
      <td valign="left">#entry#</td>
      <td valign="left">#type#</td>
      <td valign="left">#value#</td>
    </tr>
  </cfoutput>
</table>

</body>
</html>
```

Figure 33.12

Output from the `<CFREGISTRY> ACTION=GetAll` example.



Summary

ColdFusion provides several powerful tools to interact with the operating system, including the <CFFILE>, <CFDIRECTORY>, <CFEXECUTE>, and <CFREGISTRY> tags. With this power comes potential for harm to the system, so extreme caution needs to be exercised when using these tags in applications, especially if user input is used to drive the tags.