CHAPTER 32

Generating Non-HTML Content

IN THIS CHAPTER

About ColdFusion and Non-HTML Content

Normally, ColdFusion is used to generate Web pages and Web pages only. Its main purpose in life is to wait for requests from Web browsers and send back chunks of HTML in response. The Web browsers then render the HTML visually for your users. With the exception of the Flash Remoting listings in Chapter 23, HTML generation is what every code example in this book has been about.

However, there's no law that says your ColdFusion templates have to send HTML back to the Web browser. If you think about it, the only reason HTML tags such as `<body>` and `<title>` are in the pages ColdFusion serves up is because you put them there. Take away the HTML tags, and your ColdFusion templates are really just sending back plain old text files, such as you would create in Notepad or any other text editor.

Okay, so if you remove the HTML tags from a page, could you format or mark up the text in some other way? Sure you could. If the user's browser knows how to display text formatted in that particular way, it will do so. If not, it will try to invoke some other application to display the content to the user.

In this chapter, you learn about doing just that: getting ColdFusion to create other types of content on-the-fly.

About MIME Content Types

If you've been working with Web pages for long, you might have heard of strange things called MIME types or content types. But you may not have heard much of an explanation regarding what exactly they are or what they are good for.

Basically, a MIME type is a short label that determines what type of content a particular document or URL contains. The idea is that every file format—word processing files, spreadsheets, HTML files, image files, multimedia files, and so on—has (or could have) its own MIME type. Any Web browser, e-mail client, or other device can use the MIME type to determine what do to with a file (or stream of content from a server), such as showing it as a picture, interpreting it as a document, or opening up the file in some other program (such as the appropriate word processing or spreadsheet application).

NOTE

In a way, the concept of a MIME type is similar to the notion of a file extension in Windows; the file extension tells Windows which icon to display in the Windows Explorer and which program should be launched when a user double-clicks a file. MIME types are more flexible and appropriate for the Internet because they are not a Windows-specific concept. But both are simple schemes for describing what exactly a computer should expect to find in a particular file or chunk of data.

NOTE

Although purists might disagree, the terms *MIME type*, *MIME content type*, and *content type* are often used interchangeably. For purposes of this discussion, please consider them to be synonyms.

A MIME content type is always made up of two parts, separated by a forward slash:

- The first part describes the broad category that the content can be thought to belong to. The common ones are text (generally something that could sensibly be opened in Notepad or some other text editor), image, audio, video, and application. The application type is somewhat of a catchall, generally meaning that the content is meant for a specific application, such as Word, Photoshop, or something of that nature.
- The second part, or subtype, is a more specific description of what exactly the content's format is. For instance, JPEG images, GIF images, and TIFF images been given MIME types of `image/jpeg`, `image/gif`, and `image/tiff`, respectively.

Some common content types are listed in Table 32.1.

Table 32.1 Examples of Common MIME Content Types

MIME TYPE	DESCRIPTION
`text/html`	Content that should be interpreted as HTML markup and thus rendered by a Web browser natively. By far the most common in use on the Web. Anything you would normally call a Web page has this content type.
`text/plain`	Just normal text content, such as you would create if you were just typing a note for yourself in Notepad or some other text editor.
`text/vnd.wap.wml`	Markup content intended for WAP-enabled wireless devices, such as cellular phones. This subject is discussed in the section "Getting Started with Wireless Applications," later in this chapter.
`image/gif`, `image/jpeg`	Common types of image content. Unlike the various text listed above, images are binary files that mostly contain information about individual pixels and can't be opened in a text editor such as Notepad.

Table 32.1 (continued)

MIME TYPE	DESCRIPTION
`application/msword`, `application/msexcel`	Content tailored specifically for (and probably generated or saved by) a specific application, such as Microsoft Word or Excel.
`application/` `x-shockwave-flash`	Multimedia presentations or movies to be displayed by the Macromedia Flash Player.
`video/x-msvideo`, `video/vnd.rn-realvideo`	Multimedia presentations or movies to be displayed by the Windows Media Player or Real Player, respectively.
`application/unknown` or `application/octet-stream`	In general, a file or content that isn't really meant to be opened or viewed in any particular way. When you download an executable program (an .exe file, perhaps) from a Web site, the content type typically is set to this value. Web browsers usually respond to this content type by prompting the user for a download location by displaying a Save As prompt.

How Your Browser Handles MIME Types

A good way to get a list of MIME content types—and which program your particular browser will launch if it can't display the content itself—is to examine how the browser has been configured.

If you are using a Netscape or Mozilla browser, select Preferences from the Edit menu, and then select the Applications or Helper Applications option (depending on the version). You will be able to scroll through the list of applications and see the types of content the browser has been configured to launch for each application. For instance, URLs that return a content type of `application/msexcel` are passed off to Microsoft Excel for processing and viewing (Figure 32.1). This is how the dialog box looks with Netscape 4.7; other versions will look different but provide the same basic information.

If you are using Internet Explorer on a Windows machine, the equivalent place to look is in the File Types tab of the Folder Options dialog box, which you can get to by selecting Folder Options from the Tools menu in the Windows Explorer (not Internet Explorer).

NOTE

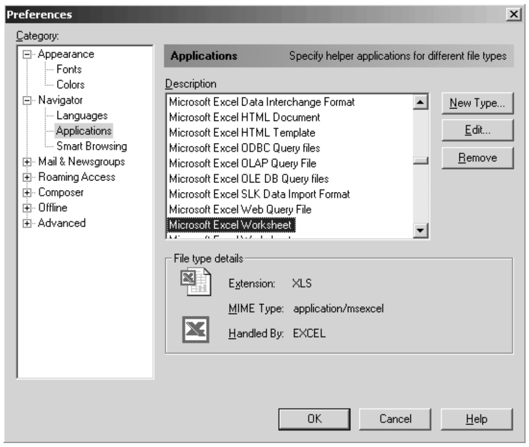
If no content type is provided by the Web server for a given file or steam of content, the browser will not know what to do with the content. It usually will ask you what you want to do with the file, often by presenting a dialog box in which you select the appropriate viewer or helper application. The exact behavior varies from browser to browser; it is generally the same behavior as when a content type of `application/unknown` or `application/octet-stream` is specified (refer to Table 32.1).

NOTE

Although it's not exactly bedtime reading, you can read the formal specification for the MIME type scheme by visiting the WC3's Web site at `http://www.w3.org`.

Figure 32.1

You can see how Netscape associates content types with applications in its Preferences dialog box.



Microsoft Excel Data Interchange Format
Microsoft Excel HTML Document
Microsoft Excel HTML Template
Microsoft Excel ODBC Query files
Microsoft Excel OLAP Query File
Microsoft Excel OLE DB Query files
Microsoft Excel SLK Data Import Format
Microsoft Excel Web Query File
Microsoft Excel Worksheet

Extension: XLS
MIME Type: application/msexcel
Handled By: EXCEL

Introducing the `<CFCONTENT>` Tag

The HTTP specification requires that every response from a Web server include the appropriate content type information. By default, ColdFusion always sends back a MIME type of `text/html`, which is why Web browsers assume that the text returned by your templates should be rendered as Web pages.

The `<CFCONTENT>` tag is provided for situations in which you want to send a different content type back to the browser along with the content your template generates. Table 32.2 lists the various attributes the `<CFCONTENT>` tag can take. The most important of these, as you might guess, is the `TYPE` attribute.

Table 32.2 `<CFCONTENT>` Tag Syntax

ATTRIBUTE	PURPOSE
`TYPE`	Required. The `MIME` content type you want to send back to the browser along with the content your template is generating. This could be one of the types listed in Table 32.1 or some other type. For instance, if your template is written to generate plain text (rather than HTML), you would set this value to `text/plain`.
`FILE`	Optional. The complete path of a file on the server that you want to send back to the browser. The actual content of the file should match up with the content type you specified with the `TYPE` attribute. You can use this to respond to a page request with an image or some other type of file that you want the user to download (perhaps a .zip or an .exe file).
`DELETEFILE`	Optional. You can set this attribute to `Yes`, which causes a file to be deleted from the server's drive after it has been sent to the browser. The default is `No`. Relevant only if the `FILE` attribute is provided.
`RESET`	Optional. Defaults to `Yes`, which means that any text that might have been output before the `<CFCONTENT>` tag should be discarded (not sent to the browser). This is handy if your code needs to make some decisions before deciding which content type to specify. If it's set to `No`, all content, even spaces and other whitespace, that precedes the `<CFCONTENT>` tag will be sent back to the browser. This attribute is discussed later in this chapter, in the section "Generating Comma-Separated Text."

Experimenting with Plain Text

As mentioned previously, ColdFusion always sends back a content type of `text/html` unless you specify a different one using the `<CFCONTENT>` tag. As an experiment, you can try setting the content type to `text/plain`, which means the browser is not obligated to parse the content as HTML. Instead, the browser can display the content literally, just as you would expect to see the content in a simple text editor such as Notepad.

Here's a simple exercise to illustrate this point. Visit Listings 32.1 and 32.2 with your Web browser, and compare the results. You can probably guess what Listing 32.1 will look like: The words `The time is now` will be in italics, and then the current time will appear in bold on the next line. Listing 32.2, on the other hand, tells the browser that the content type is plain text rather than HTML, so it shows the text quite literally, without attempting to apply any tag-based formatting (Figure 32.2).

Listing 32.1 `TestMessage1.cfm`—A Simple Message, Without Specifying a Content Type

```
<!---
  Filename: TestMessage1.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Outputting an HTML message normally
--->

<!-- Display the current time -->
<P><I>The time is now:</I><BR>
<CFOUTPUT><B>#TimeFormat(Now())#</B></CFOUTPUT>
```

Listing 32.2 `TestMessage2.cfm`—The Same Simple Message, Specifying `text/plain` as the Content Type

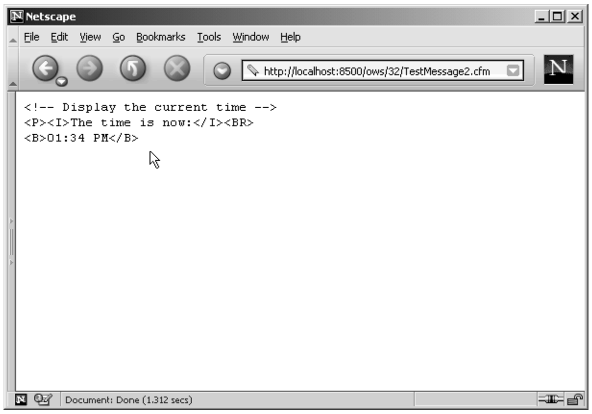
```
<!---
  Filename: TestMessage1.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Outputting an HTML message normally
--->

<CFCONTENT TYPE="text/plain">
<!-- Display the current time -->
<P><I>The time is now:</I><BR>
<CFOUTPUT><B>#TimeFormat(Now())#</B></CFOUTPUT>
```

NOTE

Unlike the other listings in this book, Listings 32.1 and 32.2 were purposefully written to not be well-formed HTML (no `<html>` or `<body>` tags and so on). This is because Internet Explorer browsers try to second-guess the situation by assuming that any document that contains well-formed HTML should be displayed as HTML, regardless of the content type specified by the Web server. Whether that's a bug or a feature depends on who you ask (see "Adding a `Content-Disposition` Header for Internet Explorer," later in this chapter). We just chose these short listings to avoid any confusion. You'll find that Netscape browsers will correctly display even a well-formed HTML document as plain text if you specify a content type of `text/plain`.

NOTE

The `TimeFormat()` and `Now()` functions are both explained in Appendix C, "ColdFusion Function Reference."



Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost:8500/ows/32/TestMessage2.cfm
Document: Done (1.312 secs)

```
<!-- Display the current time -->
<P><I>The time is now:</I><BR>
<B>01:34 PM</B>
```

Figure 32.2

If the MIME type is set to `text/plain`, the browser won't know to interpret any HTML tags in your documents.

Comma-Separated Text

Now that you know how to use ColdFusion to serve plain text rather than HTML, let's try to put that knowledge to good use. One thing people do with plain-text files is to use them to hold comma-separated text. Comma-separated text is a simple data format that gets used for many purposes. Because it's simple and easy to parse through, it's often used for various types of logging and simple integration projects. In fact, ColdFusion's own log files are kept in a comma-separated format (take a look at the files in the `LOG` folder, within ColdFusion's program directory). Many types of Web server software packages keep their logs in comma-separated format as well.

There are slightly different flavors of comma-separated text in common use, but they usually follow these rules:

- Each row of information sits in its own line in the text.
- Within each line, each column or field of information has a comma separating it from the next column or field.
- If the data in a particular column can contain commas, it's traditional to put double-quote characters around the data so the "real" commas can be distinguished from the commas that separate the fields. If the data contains actual double-quote characters, they are usually escaped by using two double-quote characters together.
- It's common to put the names of each column on the first line, with each name surrounded by quotation marks and separated from one another with commas.

Generating Comma-Separated Text

To generate comma-separated text with a ColdFusion template, you are generally going to create a .cfm file that does the following:

1. Retrieves the information that should be presented in column-separated format, using a database query or some other means.

2. Sets the content type for the request to `text/plain` using the `<CFCONTENT>` tag, so the generated content is not mistaken for HTML.
3. Outputs the names of the columns as the first line of the generated content.
4. Outputs the actual rows of data, each on its own line.

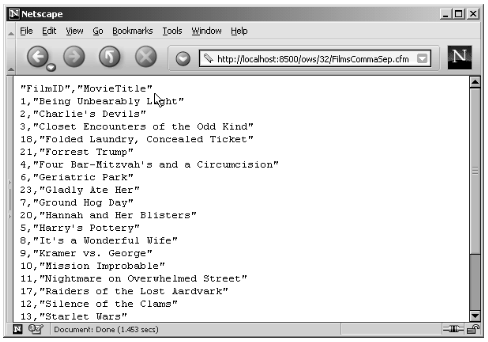
The code in Listing 32.3 retrieves some data from a database table and sends the data back to the browser in a comma-separated text format (Figure 32.3).

Listing 32.3 `FilmsCommaSep.cfm`—Presenting Queried Data As Comma-Separated Text

```
<!---
  Filename: FilmsCommaSep.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Outputs film information as comma-separated text
--->

Retrieving information about films...
<CFQUERY DATASOURCE="ows" NAME="GetFilms">
  SELECT FilmID, MovieTitle
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- Now output as simple comma-separated text --->
<!--- Put the column names on first line, then  --->
<!--- the actual data rows on their own lines   --->
<CFCONTENT TYPE="text/plain">"FilmID","MovieTitle"
<CFOUTPUT QUERY="GetFilms">#FilmID#,"#MovieTitle#"
</CFOUTPUT>
```

Figure 32.3

The `<CFCONTENT>` tag makes generating comma-separated text on-the-fly easy.



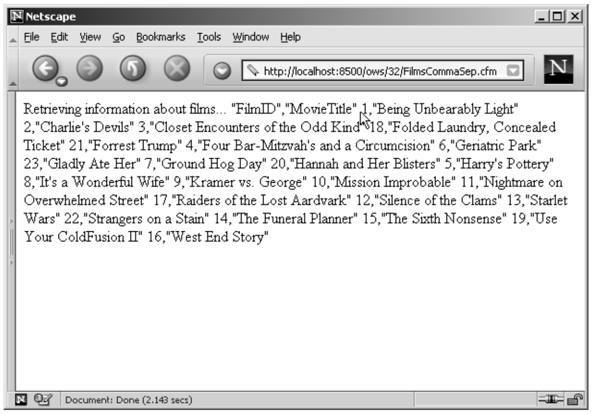
Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost:8500/ows/32/FilmsCommaSep.cfm
Document: Done (1.453 secs)

```
"FilmID","MovieTitle"
1,"Being Unbearably Light"
2,"Charlie's Devils"
3,"Closet Encounters of the Odd Kind"
18,"Folded Laundry, Concealed Ticket"
21,"Forrest Trump"
4,"Four Bar-Mitzvah's and a Circumcision"
6,"Geriatric Park"
23,"Gladly Ate Her"
7,"Ground Hog Day"
20,"Hannah and Her Blisters"
5,"Harry's Pottery"
8,"It's a Wonderful Wife"
9,"Kramer vs. George"
10,"Mission Improbable"
11,"Nightmare on Overwhelmed Street"
17,"Raiders of the Lost Aardvark"
12,"Silence of the Clams"
13,"Starlet Wars"
```

NOTE

In HTML, whitespace (such as multiple spaces together or new lines) is ignored–not so with comma-separated text. That's why there is a new line after the list of column names but not before; it is important that the names show up as the first line of text. That's also why there is a new line before the closing `</CFOUTPUT>` tag but not after the opening `<CFOUTPUT>` tag. You want ColdFusion to output a new line only after each row of data, so you can't freely indent and skip lines the way you can when you are outputting normal HTML content.

You'll notice that the `Retrieving information about films` message is not displayed by the browser. Because you have not explicitly set the `RESET` attribute of the `<CFCONTENT>` tag to `No`, it defaults to its `Yes` behavior, which is for all content that was generated before the tag to be discarded. If you were to remove the `<CFCONTENT>` tag, not only would the data no longer be displayed as plain text by the browser, but the retrieving message would be displayed, as you would expect it to normally (Figure 32.4). The same results would appear if you added `RESET="No"` to the `<CFCONTENT>` tag.


Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost:8500/ows/32/FilmsCommaSep.cfm
Document: Done (2.143 secs)

```
Retrieving information about films... "FilmID","MovieTitle" 1,"Being Unbearably Light"
2,"Charlie's Devils" 3,"Closet Encounters of the Odd Kind" 18,"Folded Laundry, Concealed
Ticket" 21,"Forrest Trump" 4,"Four Bar-Mitzvah's and a Circumcision" 6,"Geriatric Park"
23,"Gladly Ate Her" 7,"Ground Hog Day" 20,"Hannah and Her Blisters" 5,"Harry's Pottery"
8,"It's a Wonderful Wife" 9,"Kramer vs. George" 10,"Mission Improbable" 11,"Nightmare on
Overwhelmed Street" 17,"Raiders of the Lost Aardvark" 12,"Silence of the Clams" 13,"Starlet
Wars" 22,"Strangers on a Stain" 14,"The Funeral Planner" 15,"The Sixth Nonsense" 19,"Use
Your ColdFusion II" 16,"West End Story"
```

Figure 32.4

Without `<CFCONTENT>`, the new lines in the text are no longer displayed. Also, the message from earlier in the template is included in the output.

Adding a Content-Disposition Header for Internet Explorer

If you test Listing 32.3 with a Netscape browser, it should look similar to Figure 32.3. If you visit the listing with Internet Explorer 5, you might see much different behavior. For instance, you might find that the comma-separated text automatically opens in Dreamweaver as a file named `FilmsCommaSep[1].cfm` or similar. Internet Explorer takes the `.cfm` extension in the URL as an indication that the returned content is a ColdFusion template, which it clearly is not.

This illustrates one of Internet Explorer 5's policies, which, contrary to most other browsers, is to think of the file extension—and the format of the actual content itself—as being more important than the specified content type. Only if the file extension does not have an associated program—or when there is no file extension—is the content type considered.

NOTE

This behavior has changed a bit with the introduction of Internet Explorer 6. IE 6 behaves like Netscape browsers with respect to content type handling. Previous versions of IE (versions 4, 5, and 5.5) behave as discussed in this section.

To get consistent behavior with IE, add a `content-disposition` header to the page request, using a `<CFHEADER>` tag, such as the following. The `<CFHEADER>` tag is the method ColdFusion provides to enable you to send custom HTTP headers back to the browser, along with the content your template generates. You would add this line right before the `<CFCONTENT>` line in Listing 32.3:

```
<CFHEADER
  NAME="Content-Disposition"
  VALUE="filename=films">
```

We don't have space here to discuss the `content-disposition` header in detail. The short explanation is that the `content-disposition` header suggests a filename for the file, if the user were to save it. After Internet Explorer sees this header, it will think of the content as having an implied filename of `films`, rather than of `FilmsCommaSep.cfm`. Because `films` has no extension, Internet Explorer will then use the content type to decide how to display the content. The content therefore will be shown as plain text, similar to Figure 32.3.

You could also provide a filename with an extension, like so:

```
<CFHEADER
  NAME="Content-Disposition"
  VALUE="filename=films.txt">
```

Now, if you visit the listing with Internet Explorer and then select Save As from the File menu, the Save Web Page dialog box will appear with `films.txt` prefilled as the filename. Mozilla-based browsers will behave in the same way.

You can also add the word `Attachment` to the header, as shown here:

```
<CFHEADER
  NAME="Content-Disposition"
  VALUE="Attachment; filename=films.txt">
```

This will cause Internet Explorer to prompt the user to save the content as an attachment right away when Listing 32.3 is visited, rather than displaying the content in the browser window. Netscape browsers do not do anything special with the `Attachment` keyword, so the added line of code should not cause any harm. The `FilmsCommaSepIE.cfm` template (on this book's CD-ROM) includes this `content-disposition` line.

NOTE

To read more about the `content-disposition` header and the `Attachment` and `filename` keywords (there is an `Inline` keyword, too), see RFC 1806 at `http://www.ietf.org`.

NOTE

If you want to know more about the way IE decides how to display incoming content, please refer to Article number Q293336 in the Microsoft Knowledge Base at `http://support.microsoft.com`.

Retrieving the Comma-Separated Text with `<CFHTTP>`

Listing 32.4 is a bit of a digression, but it demonstrates a way you actually can use comma-separated text in your applications. You can use ColdFusion's `<CFHTTP>` tag to visit the URL for Listing 32.3, as if ColdFusion itself were a Web browser. The `<CFHTTP>` tag grabs the comma-separated text generated by `FilmsCommaSep.cfm` and parses through it, effectively re-creating the query result set. You can then use the resultset just as if it were returned by the original `<CFQUERY>` tag. This enables you to share data between two different ColdFusion servers over the Internet.

NOTE

We're using the `<CFHTTP>` tag a bit ahead of ourselves here. It's discussed in full in the "Using Server-Side HTTP and FTP" chapter in our companion volume, Advanced ColdFusion MX Application Development (Peachpit). You can find the syntax for the `<CFHTTP>` tag in Appendix B, "ColdFusion Tag Reference".

NOTE

ColdFusion's `<CFWDDX>` tag gives you an even easier and more powerful way to share content between servers over the Internet, using a clever XML format instead of comma-separated text. Unfortunately, that discussion is beyond the scope of this book. To learn more about WDDX, please refer to the "Using WDDX" chapter in Advanced ColdFusion MX Application Development or visit `http://www.openwddx.org`.

Listing 32.4 `FetchCommaSep.cfm`—The `<CFHTTP>` Tag Knows How to Grab Comma-Separated Text Over the Internet and Return It to You as a Query Object

```
<!---
  Filename: FetchCommaSep.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Retrieves comma-separated film information over the Web
--->

<!--- Visit our new "comma-separated" page, --->
<!--- and parse content into a query object --->
<CFHTTP
  METHOD="Get"
  URL="http://localhost:8500/ows/32/FilmsCommaSep.cfm"
  NAME="GetFilmsViaHTTP"
  DELIMITER=",">

<HTML>
<HEAD><TITLE>Fetching Comma-Separated Content</TITLE></HEAD>
<BODY>

<H3>Comma-Separated text has been fetched via CFHTTP</H3>

<!--- Now the "fetched" query may be used normally --->
<CFOUTPUT QUERY="GetFilmsViaHTTP">
  Film <B>#FilmID#</B> is: <I>#MovieTitle#</I><BR>
</CFOUTPUT>

</BODY>
</HTML>
```

Generating Excel Files

You might have seen Web sites that enable users to access or view some type of information in spreadsheet or Excel format. For instance, the check stock quotes area at Yahoo! enables users to download stock quotes in spreadsheet format. Because Microsoft Excel has become the de facto standard for spreadsheet applications, and because so many PCs come with it preinstalled, this type of feature can be very useful for people.

In this section, you learn how you can add this type of functionality to your own ColdFusion applications. This will enable you to provide your users with a way to open customized data in Excel (such as kind of personalized purchase history or a list of other employees if you're building an intranet application).

Creating Spreadsheets with Tab-Separated Text

One of the file formats Excel knows how to import is tab-separated text. Tab-separated text is basically the same as comma-separated text, except it uses tab characters instead of commas to separate the columns on each line.

If you create a ColdFusion template that generates tab-separated text and specify a content type of `application/msexcel`, the user's browser will display the content to the user by launching Excel. Excel should then seamlessly import the tab-separated text and display it in the same way it would display a native Excel worksheet (.xls) file. As far as the end user is concerned, your Web site presented him with a personalized spreadsheet file. Only you know that it was simply some tab-separated text.

Listing 32.5 shows how easy it is to get this effect. Note that this code is essentially the same as the code from Listing 32.3, with just a few minor changes. Most importantly, the `TYPE` attribute of the `<CFCONTENT>` tag has been changed to `application/msexcel`.

Listing 32.5 `FilmsToExcel.cfm`—Generating Tab-Separated Text for Use with Excel on the Client

```
<!---
  Filename: FilmsToExcel.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Outputs film information for Microsoft Excel
--->

<!--- Don't output anything *not* in CFOUTPUT tags --->
<!--- This makes it easier to deal with whitespace --->
<CFSETTING ENABLECFOUTPUTONLY="Yes">

<!--- Retrieve information about films --->
<CFQUERY DATASOURCE="ows" NAME="GetFilms">
  SELECT MovieTitle, AmountBudgeted
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- Set variables for special characters --->
<CFSET TabChar = Chr(9)>
<CFSET NewLine = Chr(13) & Chr(10)>

<!--- Suggest default filename for spreadsheet --->
<CFHEADER NAME="Content-Disposition" VALUE="filename=FilmBudgets.xls">

<!--- Set the content-type so Excel is invoked --->
<CFCONTENT TYPE="application/msexcel">

<!--- Output the header row, with column names --->
<!--- Put tab between columns, and newline at end --->
<CFOUTPUT>MOVIE TITLE#TabChar#BUDGET#NewLine#</CFOUTPUT>

<!--- Output actual data rows, each on own line --->
<!--- Put tab between columns, and newline at end  --->
<CFLOOP QUERY="GetFilms">
  <CFOUTPUT>#MovieTitle##TabChar##AmountBudgeted##NewLine#</CFOUTPUT>
</CFLOOP>
```

NOTE

When the spreadsheet is displayed in Excel, you might see hash marks (a series of # signs) where the budget numbers belong. Just make the budget column a bit wider (by dragging in Excel) to see the full numbers. This is normal Excel behavior and does not represent a problem or bug.

Another difference is that this code uses the `<CFHEADER>` tag mentioned earlier to send a custom `Content-Disposition` header to the browser, along with the content type. The `Content-Disposition` header is used to suggest a default filename for the content going back to the browser (see the following notes and the section, "Adding a `Content-Disposition` Header for Internet Explorer," earlier in this chapter).

NOTE

A complete discussion of all HTTP headers and the various things they are used for is beyond the scope of this book (and is something you generally don't need to know about because ColdFusion takes care of this kind of thing for you). In short, custom headers can be used to provide the browser with various pieces of information, or metadata, about the server's response.

NOTE

The `Content-Disposition` header really should be optional, but because of the way Internet Explorer determines how to handle incoming content, this particular `<CFHEADER>` tag is required for the code to work properly with IE. To make the decision about which application to launch, IE doesn't look only at the MIME content type the code provides; it also relies on other factors, such as the suggested filename. Additionally, the extension of the suggested filename must be associated with Excel (that is, .xls), but the first part of the filename is up to you. If you want to know more about the way IE decides how to display incoming content, please refer to Article number Q293336 in the Microsoft Knowledge Base at `http://support.microsoft.com`.

Listing 32.5 also sets a couple of variables for the two special characters needed to produce the correct text, by using the `Chr()` function. This can be a more manageable way to get special characters into your page output. Instead of actually pressing the Tab key to insert a Tab character into your code (which would work but might be hard to notice or understand when you edit the code later), you can use the `Chr()` function, which returns the character specified by the ASCII code you supply.

NOTE

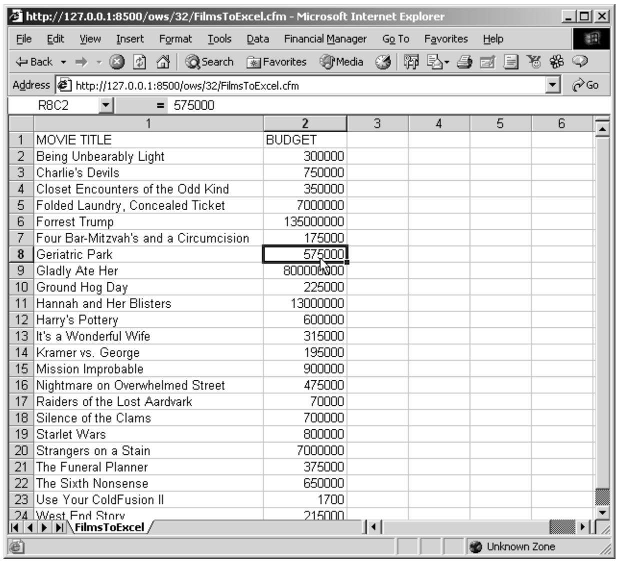
`Chr(9)` always returns a Tab character, and a `Chr(13)` followed by a `Chr(10)` always returns a linefeed character followed by a carriage return. A linefeed followed by a carriage return is known as a newline, which is the standard way to indicate the end of a line in a text file.

Another difference is that this listing turns on the `ENABLECFOUTPUTONLY` mode of the `<CFSETTING>` tag, which causes ColdFusion not to output anything that is not between `CFOUTPUT` tags. Together with the `TabChar` and `NewLine` variables, this enables the code to explicitly tell ColdFusion about every single character that needs to be output to the browser, rather than having to be overly careful about positioning the tags within the code (refer to Listing 32.3). See Appendix B for more information about the `<CFSETTING>` tag.

In any case, if you visit the URL for Listing 32.5 with your browser, it should launch Excel with the film data loaded as a spreadsheet. Depending on the browser and platform being used, Excel might be launched as a separate application, or it might appear within the browser window (Figure 32.5).

Figure 32.5

If you send tab-separated text back to the browser with the correct content type, it should be opened in Excel.



http://127.0.0.1:8500/ows/32/FilmsToExcel.cfm - Microsoft Internet Explorer

Address http://127.0.0.1:8500/ows/32/FilmsToExcel.cfm

R8C2 = 575000

	1	2	3	4	5	6
1	MOVIE TITLE	BUDGET				
2	Being Unbearably Light	300000				
3	Charlie's Devils	750000				
4	Closet Encounters of the Odd Kind	350000				
5	Folded Laundry, Concealed Ticket	7000000				
6	Forrest Trump	135000000				
7	Four Bar-Mitzvah's and a Circumcision	175000				
8	Geriatric Park	575000				
9	Gladly Ate Her	800000000				
10	Ground Hog Day	225000				
11	Hannah and Her Blisters	13000000				
12	Harry's Pottery	600000				
13	It's a Wonderful Wife	315000				
14	Kramer vs. George	195000				
15	Mission Improbable	900000				
16	Nightmare on Overwhelmed Street	475000				
17	Raiders of the Lost Aardvark	70000				
18	Silence of the Clams	700000				
19	Starlet Wars	800000				
20	Strangers on a Stain	7000000				
21	The Funeral Planner	375000				
22	The Sixth Nonsense	650000				
23	Use Your ColdFusion II	1700				
24	West End Story	215000				

NOTE

Of course, you could create a template based on Listing 32.5 that uses a URL parameter to dynamically return spreadsheet data based on some type of Film ID, actor, and so on.

Creating Spreadsheets with HTML

For Excel 2000, Microsoft created a special flavor of HTML, which Excel can import as if it were a normal .xls file. If you have Excel 2000, Excel XP, or later, try creating a quick spreadsheet with a few columns and rows. Select Save As from the File menu, and save the spreadsheet as an HTML file. Now open that HTML file in a text editor such as Notepad or Dreamweaver MX. You will find a decent amount of what seems like extraneous code in there, but after a moment you'll realize that the spreadsheet has basically just been converted into an ordinary HTML table, using the `<table>`, `<tr>`, and `<td>` tags you are already familiar with.

If you were to create a ColdFusion template that created a similar HTML file, Excel could render it as a spreadsheet, just as it could render the tab-separated text from Listing 32.5 as a spreadsheet. The advantage to using the special HTML format over the tab-separated format is that Microsoft designed the special HTML format to be more than just a data-export format; it actually holds all the Excel-specific information about the spreadsheet, as well. This means you can dynamically specify formatting options such as font, color, and alignment, and even provide autocalculating formulas for specific cells. So, you easily can create a fully functioning Excel spreadsheet on-the-fly, using relatively familiar HTML-looking syntax.

Listing 32.6 and Listing 32.7 are two examples that demonstrate how you can create formatted spreadsheets for Excel by using HTML table syntax. Listing 32.6 again creates a spreadsheet that lists the title and budget for each film. Note that you can provide width and alignment for the columns and specify formatting using ordinary `<font>` and `<b>` tags.

Listing 32.6 `FilmsToExcelPretty.cfm`—Outputting an HTML Table for Display in Excel

```
<!---
  Filename: FilmsToPretty.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Outputs film information for Microsoft Excel
--->

<!--- Retrieve information about films --->
<CFQUERY DATASOURCE="ows" NAME="GetFilms">
  SELECT MovieTitle, AmountBudgeted
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- Set the content-type so Excel is invoked --->
<CFCONTENT TYPE="application/msexcel">

<!--- Suggest default filename for spreadsheet --->
<CFHEADER NAME="Content-Disposition" VALUE="filename=FilmBudgets.xls">

<html>
<head><title>Film Budgets</title></head>
<body>

<!--- Output ordinary HTML table, which will --->
<!--- be displayed by Excel as a spreadsheet --->
<table>
  <tr><th>Film</th><th>Budget</th></tr>
  <CFOUTPUT QUERY="GetFilms">
    <tr>
      <td width="400">
        <font face="verdana">#MovieTitle#</font>
      </td>
      <td align="center">
        <font color="red"><b>#AmountBudgeted#</b></font>
      </td>
    </tr>
  </CFOUTPUT>
</table>

</body>
</html>
```

Listing 32.7 is similar to Listing 32.6, except that it includes some additional color and formatting instructions, so the resulting spreadsheet looks quite nice. It also adds another row of cells to the bottom of the spreadsheet, which shows the total of the second column (the budgets of all films combined). This is a live, formula-based total; if the user changes any of the prices, Excel updates the total accordingly (Figure 32.6).

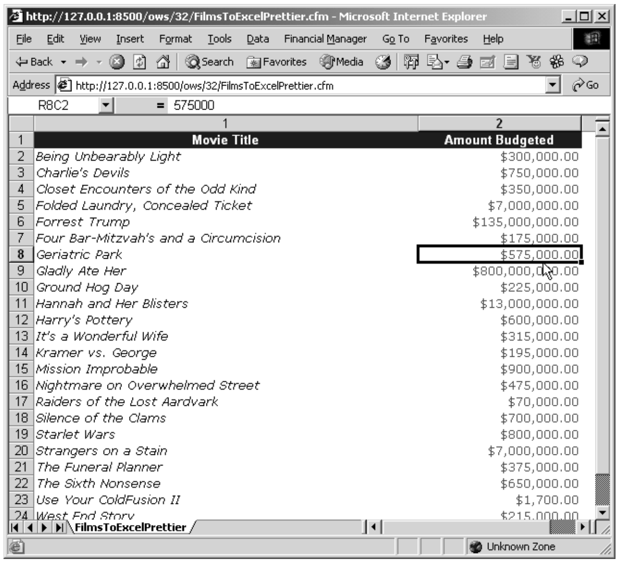


Figure 32.6

You can create dynamic spreadsheets that include live cell formulas.

http://127.0.0.1:8500/ows/32/FilmsToExcelPrettier.cfm - Microsoft Internet Explorer
File Edit View Insert Format Tools Data Financial Manager Go To Favorites Help
Back Search Favorites Media
Address http://127.0.0.1:8500/ows/32/FilmsToExcelPrettier.cfm
Go
R8C2
= 575000
FilmsToExcelPrettier
Unknown Zone

	1	2
1	**Movie Title**	**Amount Budgeted**
2	*Being Unbearably Light*	$300,000.00
3	*Charlie's Devils*	$750,000.00
4	*Closet Encounters of the Odd Kind*	$350,000.00
5	*Folded Laundry, Concealed Ticket*	$7,000,000.00
6	*Forrest Trump*	$135,000,000.00
7	*Four Bar-Mitzvah's and a Circumcision*	$175,000.00
8	*Geriatric Park*	$575,000.00
9	*Gladly Ate Her*	$800,000,000.00
10	*Ground Hog Day*	$225,000.00
11	*Hannah and Her Blisters*	$13,000,000.00
12	*Harry's Pottery*	$600,000.00
13	*It's a Wonderful Wife*	$315,000.00
14	*Kramer vs. George*	$195,000.00
15	*Mission Improbable*	$900,000.00
16	*Nightmare on Overwhelmed Street*	$475,000.00
17	*Raiders of the Lost Aardvark*	$70,000.00
18	*Silence of the Clams*	$700,000.00
19	*Starlet Wars*	$800,000.00
20	*Strangers on a Stain*	$7,000,000.00
21	*The Funeral Planner*	$375,000.00
22	*The Sixth Nonsense*	$650,000.00
23	*Use Your ColdFusion II*	$1,700.00
24	*West End Story*	$215,000.00

Listing 32.7 `FilmsToExcelPrettier.cfm`—Adding CSS Formatting and Cell Formulas to the Generated Spreadsheet

```
<!---
  Filename: FilmsToExcelPrettier.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Outputs film information for Microsoft Excel
--->

<!--- Retrieve information about films --->
<CFQUERY DATASOURCE="ows" NAME="GetFilms">
  SELECT MovieTitle, AmountBudgeted
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- Set the content-type so Excel is invoked --->
<CFCONTENT TYPE="application/msexcel">

<!--- Suggest default filename for spreadsheet --->
<CFHEADER NAME="Content-Disposition" VALUE="filename=FilmBudgets.xls">

<!--- Include "XML Namespace" information to --->
<!--- allow using Excel "extensions" to HTML --->
<html
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  xmlns="http://www.w3.org/TR/REC-html40">
<head><title>Film Budgets</title></head>
```

Listing 32.7 (continued)

```
<body>

<style TYPE="text/css">
  .rowHeads {
    color:white;
    background:blue;
  }
  .titleCol {
    width:400px;
    font-style:italic;
    font-family:verdana;
  }
  .priceCol {
    width:170px;
    font-family:verdana;
    color:red;
    mso-number-format:"\0022$\0022\#\,\#\#0\.00"
  };
</style>


<!--- Output ordinary HTML table, which will --->
<!--- be displayed by Excel as a spreadsheet --->
<table>
  <!--- Top row --->
  <tr>
    <th class="rowHeads">Movie Title</th>
    <th class="rowHeads">Amount Budgeted</th>
  </tr>

  <!--- Data rows --->
  <CFOUTPUT QUERY="GetFilms">
    <tr>
      <td class="titleCol">#MovieTitle#</td>
      <td class="priceCol">#AmountBudgeted#</td>
    </tr>
  </CFOUTPUT>

  <!--- Last row, with "total" formula --->
  <CFSET FirstPriceCell = "B2">
  <CFSET LastPriceCell  = "B" & GetFilms.RecordCount + 1>
  <CFSET TotalFormula   = "SUM(#FirstPriceCell#:#LastPriceCell#)">
  <CFOUTPUT>
    <tr>
      <td
        class="titleCol"
        style="font-weight:bold;background:yellow">Total:</td>
      <td
        class="priceCol"
        style="font-weight:bold;background:yellow"
        x:fmla="=#TotalFormula#"></td>
    </tr>
  </CFOUTPUT>
</table>

</body>
</html>
```

Near the end of Listing 32.7, a ColdFusion variable called `TotalFormula` is created, which will end up having a value of `SUM(B2:B24)` if 23 rows of film information exist. That formula is then supplied to the special `x:fla` attribute of the `<td>` tag, which is one of the Microsoft extensions to HTML geared especially for use with Excel. To use these extensions, the three `xmlns` attributes must be included for the `<html>` tag at the top of the template.

NOTE

We couldn't possibly explain everything about these special extensions to HTML (and how they are implemented using XML standards) in the space we have in this book. For purposes of putting together nice-looking and functional spreadsheets for Excel, you will need to experiment a bit to figure out how to express the specific formatting or features you want using HTML. A good way to do this is to simply create a similar spreadsheet using Excel normally; then save the spreadsheet as HTML, as explained earlier in this section. By examining the HTML file in a text editor, you will be able to figure out how to get the results you want on-the-fly.

NOTE

Remember that the HTML approach used in Listing 32.7 is a solution only for Excel 2000 (sometimes called Excel 9) and later. This technique will not work for earlier versions of the product.

Other Options for Creating Excel Files

A few other options are available with regard to generating Excel spreadsheets dynamically with ColdFusion. Unfortunately, there isn't space to go into a specific discussion about each of these options here.

Creating Files Server-Side

Instead of generating a new Excel spreadsheet on-the-fly for every request, you could write the necessary tab-delimited or HTML content to a file on the server, using the `<CFFILE>` tag. Then, you could send that file to the browser in response to successive page requests, using the `FILE` attribute of the `<CFCONTENT>` tag. You still would need the `<CFHEADER>` tag, as shown in Listing 32.7. The file could be updated on a periodic basis (perhaps once a day) using the `<CFSCHEDULE>` tag.

For more information about `<CFFILE>`, see Chapter 33, "Interacting with the Operating System." For more information about `<CFSCHEDULE>`, see Chapter 35, "Event Scheduling."

Talking to Excel via `<CFOBJECT>`

ColdFusion can also communicate directly with a copy of Excel installed on the server by using COM automation. You use the `<CFOBJECT>` tag with the `TYPE` attribute set to `COM` and the `CLASS` attribute set to `Excel.Application`. You then use the objects, methods, and properties provided by Excel (the same ones people normally use to create Excel macros, generally using the Visual Basic for Applications (VBA) language) to create whatever spreadsheet you want, and save it to the server's drive as a temporary file. You then can use the `FILE` attribute of the `<CFCONTENT>` tag to send the file back to the browser, perhaps also specifying a `DELETEFILE` attribute of `Yes` so the temporary file is deleted from the server after it is delivered to the browser.

TIP

Two Web sites that might help you to find additional information about creating Excel content via COM/ActiveX are `http://www.cfcomet.com` and `http://www.softartisans.com`.

For more information about `<CFOBJECT>`, please consult our Advanced ColdFusion MX Application Development book or Appendix B in this book. For the Excel-specific objects, methods, and properties mentioned previously, see your Excel documentation, the Microsoft Web site, or a third-party book about writing Excel macros using VBA.

Generating Word Files

You can also use ColdFusion to create Microsoft Word files on-the-fly, using techniques similar to the ones for creating Excel files (see the previous section). This opens up the possibility of creating personalized or customized sales documents, pricing sheets, product documentation, and other documents you might want to deliver in the common Microsoft Word file format.

Although the idea of using a proprietary document format to deliver such documents—rather than something more open, such as HTML—might rub some people the wrong way, there are some clear benefits. Most obviously, you can produce documents that the end user can edit further, using her own, familiar copy of Word. Also, in a Word document you generally have much greater control over the way the document will look when printed (margins, headers, footers, leading, kerning, widow and orphan paragraphs, and so on) than you do with HTML.

Creating Documents with RTF

If you have needed to exchange documents between word-processing programs, you might have run into a file format called the Rich Text Format (RTF). RTF is a somewhat older file format, designed to be a reasonably generic way to store formatted information, especially documents such as those you create with a word processor. Word lets you work with RTF (.rtf) files in almost all the same ways you can work with real Word (.doc) files.

Most importantly to you, RTF is a plain-text format, so it is made up of normal ASCII characters, which ColdFusion is good at generating. So, just as you were able to create a tab-separated text to be sent to Excel via `<CFCONTENT>`, you can create RTF files to be sent to Word.

Creating a Template Document

There are several approaches you could take to creating RTF files with ColdFusion. This section discusses a relatively simple one, which is similar to performing a mail merge operation within a word processor. First, you will create a template document and save it as an RTF file on your ColdFusion server. Then, you will customize the document using simple string-manipulation functions and send the customized version of the document back to the browser using the `<CFCONTENT>` tag.

NOTE

The following instructions assume you are using a recent version of Word, such as Word 2000 or Word XP. Earlier versions will also work just fine, although the specific steps you take to save a document as RTF might be slightly different.

To create your template document, do the following:

1. Open Word. Create some type of document, such as a form letter.
2. In the document, insert six placeholders by including the following terms somewhere in the text, including the percent signs: `%CurrentDate%`, `%NameFirst%`, `%NameLast%`, `%NameFirstReal%`, `%NameLastReal%`, and %`FreePrize%`.
3. Feel free to add some formatting to the document, using whatever features you want (styles, margins, font colors, tables, and so on). Just try to stay away from including large pictures in the document for the moment.
4. Select Save As from Word's File menu, select Rich Text Format from the drop-down list of file types, and save the document as `DocTemplate.rtf` in the same folder you're using to save code for this chapter.
5. Be sure you close the document in Word (otherwise, Word might keep it locked).

NOTE

If you prefer, you can just use the `DocTemplate.rtf` file included on this book's CD-ROM.

Now you can create a ColdFusion template that creates a personalized copy of the template, based on information you retrieve with a database query. Listing 32.8 shows one way to get this done. As you can see, this template has two sections. When you visit the template normally (with no URL parameters), the top portion of the code executes and displays a list of actors. The user can then click an actor's name to cause the template to be executed again, this time with the appropriate ActorID passed as a URL parameter.

Listing 32.8 `RetireActor.cfm`—Producing a Personalized Word Document from an RTF Template

```
<!---
  Filename: RetireActor.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Generates a retirement memo as an RTF document for Microsoft Word
--->

<!--- If no Actor ID passed, diplay list of links --->
<CFIF IsDefined("URL.ActorID") EQ False>
  <!--- Get a list of actors from database --->
  <CFQUERY NAME="GetActors" DATASOURCE="ows">
    SELECT ActorID, NameFirst, NameLast
    FROM Actors
    ORDER BY NameLast, NameFirst
  </CFQUERY>
```

Listing 32.8 (continued)

```
      <!--- Page Title, etc --->
      <html>
      <head><title>Actor Retirement System</title></head>
      <body>
      <h3>Which Actor Would You Like To Retire?</h3>

      <!--- For each Actor, include simple link to --->
      <!--- this page, passing the Actor ID in URL --->
      <CFOUTPUT QUERY="GetActors">
        <CFSET LinkURL = "#CGI.SCRIPT_NAME#?ActorID=#ActorID#">
        <a href="#LinkURL#">#NameFirst# <B>#NameLast#</B></a><br>
      </CFOUTPUT>

      </body>
      </html>


    <!--- If Actor ID passed, generate Word doc --->
    <CFELSE>
      <!--- Make sure Actor ID in URL is a number --->
      <CFPARAM NAME="URL.ActorID" TYPE="numeric">

      <!--- Get this Actor's name from database --->
      <CFQUERY NAME="GetActor" DATASOURCE="ows">
        SELECT NameFirst, NameLast,
          NameFirstReal, NameLastReal,
          (SELECT Min(DateInTheaters)
           FROM Films f, FilmsActors fa
           WHERE fa.FilmID = f.FilmID
           AND fa.ActorID = a.ActorID) AS DateFirstFilm
        FROM Actors a
        WHERE ActorID = #URL.ActorID#
      </CFQUERY>

      <!--- How long has Actor been with company? --->
      <CFSET MonthsEmployed = DateDiff("m", GetActor.DateFirstFilm, Now())>

      <!--- Determine severance package --->
      <CFIF MonthsEmployed GTE 36>
        <CFSET SevPackage = "Gold Watch (Digital)">
      <CFELSEIF MonthsEmployed GTE 18>
        <CFSET SevPackage = "$100 Starbucks Gift Certificate">
      <CFELSE>
        <CFSET SevPackage = "ColdFusion Web Application Construction Kit">
      </CFIF>


      <!--- Location of our RTF "template" document --->
      <CFSET ThisFolder   = GetDirectoryFromPath(GetCurrentTemplatePath())>
      <CFSET TemplatePath = ThisFolder & "DocTemplate.rtf">

      <!--- Read RTF template into variable called "RTF" --->
      <CFFILE
        ACTION="Read"
        FILE="#TemplatePath#"
```

Listing 32.8 (continued)

```
        VARIABLE="RTF">

      <!--- Replace "placeholders" with specific information --->
      <CFSET TodaysDate = DateFormat(Now(), "dddd, mmmm d, yyyy")>
      <CFSET RTF = Replace(RTF, "%CurrentDate%", TodaysDate)>
      <CFSET RTF = Replace(RTF, "%FreePrize%", SevPackage)>
      <CFSET RTF = Replace(RTF, "%NameFirst%", GetActor.NameFirst)>
      <CFSET RTF = Replace(RTF, "%NameLast%",  GetActor.NameLast )>
      <CFSET RTF = Replace(RTF, "%NameFirstReal%", GetActor.NameFirstReal)>
      <CFSET RTF = Replace(RTF, "%NameLastReal%", GetActor.NameLastReal )>

      <!--- Suggest default filename for document --->
      <CFHEADER NAME="Content-Disposition" VALUE="filename=RetireMemo.doc">

      <!--- Set the content-type so Word is invoked --->
      <CFCONTENT TYPE="application/msword"><CFOUTPUT>#RTF#</CFOUTPUT>

    </CFIF>
```

When an actor ID is provided, the second part of the template executes, which is where the dynamic Word generation occurs. First, some basic information about the employee is retrieved from the database, using an ordinary `<CFQUERY>` tag. Then, a severance package is determined, depending on how long ago the actor's first film was released (if he was hired more than 12 months ago, he gets a gold watch).

Next, the `<CFFILE>` tag is used to read the `DocTemplate.rtf` file (which is just a text file) into an ordinary ColdFusion string variable called `RTF`. Now, you can use ColdFusion's `Replace()` function to replace the simple placeholders in the RTF file with the actor's actual first and last names and replace the `%FreePrize%` placeholder with the value of the `SevPackage` variable (which contains the determined severance package).

NOTE

See Chapter 33 for more information about using `<CFFILE>` to read in the contents of text files.

NOTE

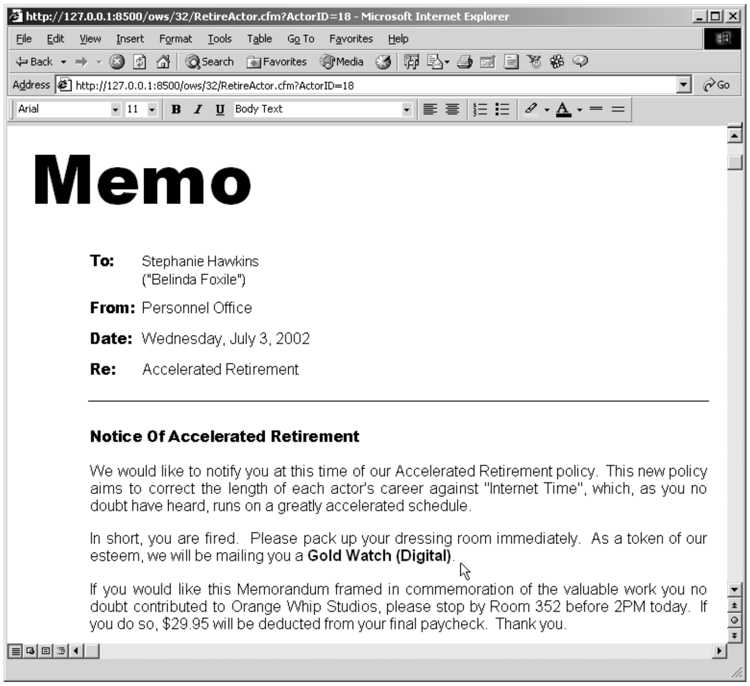
In this example, ColdFusion looks for the `DocTemplate.rtf` file in the same folder the `RetireActor.cfm` file is in. The `GetCurrentTemplatePath()` function makes this possible. In practice, you might want to keep the RTF template somewhere else, outside the Web server's document root, so it can't be downloaded directly.

Now the customized RTF text is sitting in the `RTF` variable. All that's left is to stream the RTF text to the browser. As with the Excel examples earlier in this chapter, this is done by including a `<CFCONTENT>` tag—this time with the `TYPE` attribute set to `application/msword`. This causes the browser to launch the user's copy of Microsoft Word on the local machine. Note that you also must include the `<CFHEADER>` tag, which suggests a sensible default filename for the document (see the discussion of Listing 32.5).

If you use the `DocTemplate.rtf` file that was included on this book's CD-ROM, the resulting Word document will be a personalized retirement memo (Figure 32.7).

Figure 32.7

You can create Word files on-the-fly that the user can print or modify.


http://127.0.0.1:8500/ows/32/RetireActor.cfm?ActorID=18 - Microsoft Internet Explorer
File Edit View Insert Format Tools Table Go To Favorites Help
Back Search Favorites Media
Address http://127.0.0.1:8500/ows/32/RetireActor.cfm?ActorID=18
Go
Arial 11 B I U Body Text

Memo

To: Stephanie Hawkins
("Belinda Foxile")

From: Personnel Office

Date: Wednesday, July 3, 2002

Re: Accelerated Retirement

Notice Of Accelerated Retirement

We would like to notify you at this time of our Accelerated Retirement policy. This new policy aims to correct the length of each actor's career against "Internet Time", which, as you no doubt have heard, runs on a greatly accelerated schedule.

In short, you are fired. Please pack up your dressing room immediately. As a token of our esteem, we will be mailing you a **Gold Watch (Digital)**.

If you would like this Memorandum framed in commemoration of the valuable work you no doubt contributed to Orange Whip Studios, please stop by Room 352 before 2PM today. If you do so, $29.95 will be deducted from your final paycheck. Thank you.

Other Options for Creating Word Files

A few other options are available if you want to learn more about creating Word documents with ColdFusion:

- Similar to Excel 2000, Word 2000 understands an extended version of HTML that can include extra code to deal with all the Word-specific features not normally found in HTML documents. You could adapt the basic technique described in the "Creating Spreadsheets with HTML" section of this chapter to generate the Word version of HTML.
- It should also be possible to communicate with a copy of Word installed on the ColdFusion server via the `<CFOBJECT>` tag. See "Other Options for Creating Excel Files," earlier in this chapter, for a few pointers.

Serving Media Files

You have seen how you can create various types of document-type files with a little help from the `<CFCONTENT>` tag. You also can use `<CFCONTENT>` to serve up images or other types of multimedia files, such as audio or video.

Turning ColdFusion into an Image Server

For instance, say you have been asked to create a banner ad server application with ColdFusion. One of the requirements is that your ad server must be capable of working for sites built with static HTML files. You need to be able to provide a simple `<IMG>` tag people can cut and paste into their pages. But, if the `SRC` of the `<IMG>` tag doesn't change each time a user views a page, how will the banner ad rotate? Similar to the other solutions discussed in this chapter, the answer to this head-scratcher involves the `<CFCONTENT>` tag.

Although it's certainly the norm, there is no law that says the `SRC` for an image has to have a `.gif` or `.jpg` extension. The `SRC` can be the URL for a ColdFusion template, perhaps called `AdServer.cfm`. Rather than returning some type of text, such as HTML code, the template should return an image. In addition, it should specify a MIME content type of `image/gif` instead of the usual `text/html` content type. Because of the content type, the browser will understand that it is receiving what it expects to receive (an image) and should have no problems displaying it.

Serving Up Images

Listing 32.9 shows what this ad server template might look like. It assumes that a number of banner images are located in a subfolder called `Ads`. It selects one of the ads at random and then uses `<CFCONTENT>` to send the image back to the browser with the appropriate content type.

Before this example will work, however, you must do the following:

1. Save the code shown in Listing 32.9 to a ColdFusion template named `AdServer.cfm`.
2. Create a subdirectory called `Ads`, within the same folder you just saved `AdServer.cfm` in.
3. Place several banner ads or other GIF files in the `Ads` subfolder. If you want, you can simply copy the images from the `Ads` subfolder from this chapter's folder on the CD-ROM that accompanies this book.

Listing 32.9 `AdServer1.cfm`—Using `<CFCONTENT>` to Respond with an Image Instead of HTML Content

```
<!---
  Filename:     AdServer.cfm
  Created by:   Nate Weiss (NMW)
  Purpose:      Responds with the binary content of a randomly selected image
--->

<!--- Ad Images are in "Ads" subfolder, within current folder --->
<CFSET AdDir = GetDirectoryFromPath(GetCurrentTemplatePath()) & "Ads\">

<!--- Get a listing of GIF image files in the Ads folder --->
<!--- Result is a query object with a row for each image --->
<CFDIRECTORY
  DIRECTORY="#AdDir#"
  ACTION="LIST"
```

Listing 32.9 (CONTINUED)

```
  FILTER="*.gif"
  NAME="GetAds">

<!--- Pick random number between one and number of images --->
<CFSET AdNum  = RandRange(1, GetAds.RecordCount)>
<!--- Grab the filename in chosen row of the GetAds query --->
<CFSET AdFileName = GetAds.Name[AdNum]>
<!--- Prepend directory to get full filesystem path to ad image --->
<CFSET AdFilePath = AdDir & AdFileName>

<!--- Send the chosen image back to the client --->
<CFCONTENT TYPE="image/gif" FILE="#AdFilePath#">
```

NOTE

Because it is not returning HTML content, Listing 32.9 doesn't contain the `<HTML>`, `<BODY>`, and other standard tags you're used to seeing.

As you can see, you don't need a lot of code to serve up an image with ColdFusion. You simply select the file you want to serve and supply its complete filename to the `FILE` attribute of the `<CFCONTENT>` tag, which is being used for the first time in this chapter (see Table 32.2). Any streaming and network communication issues are handled for you.

First, the `AdDir` variable is set to the full filesystem path of the `Ads` folder. This is a nice use for the `GetDirectoryFromPath()` and `GetCurrentTemplatePath()` functions. When you use the two together as shown in Listing 32.9, they always return the full path to the folder in which the currently executing template is located.

Next, the `<CFDIRECTORY>` tag is used to get a listing of all GIF files within the `Ads` folder. You can find out more about `<CFDIRECTORY>` in Chapter 33. Here, it is used in a very basic way. It returns a query object, such as the results of a `<CFQUERY>`, where each row of the query represents a file that matches the `FILTER` attribute. The query object contains a column called `NAME`, which contains the filename (with extension, but not the path) of each file.

Now, the `RandRange()` function is used to pick a number between 1 and the number of rows in the query, which in turn is the number of GIF files in the `Ads` folder. So, if eight images are in the folder, the `AdNum` variable will be set to a random number between 1 and 8. Next, the `AdFileName` variable is set to the value of the `NAME` column of the appropriate row of the `GetAds` query object. If `AdNum` turns out to be `5`, `AdFileName` is set to the filename in the fifth row of the query.

NOTE

You can gain more control over how random numbers are generated by using the `Rand()` and `Randomize()` functions together. See Appendix C for details.

Finally, another variable called `AdFilePath` is created, which just concatenates the `AdDir` and `AdFileName` variables together, resulting in the complete filesystem path to the randomly selected image file. All that's left is to feed the path to the `FILE` attribute of the `<CFCONTENT>` tag.

You should be able to test the template at this point by visiting the `AdServer.cfm` template with your browser. All you should see is the randomly selected banner ad. If you reload the template's URL a few times, you should see that the ads rotate on a fairly even, random basis.

NOTE

If you visit the `AdServer.cfm` template and then try to View Source with your browser, you will find that your browser either prevents it or shows garbage characters. This is because there is no HTML source to show. The template is returning the binary image information itself, not an `<IMG>` tag, which is a pointer to an image on the Net.

Displaying the Images in Other Web Pages

The goal of this exercise was to use a simple `<IMG>` tag to create a rotating banner ad system that could be used by static HTML files—and that's exactly what you have. You can include the following line of code in a static HTML file:

```
<IMG SRC="http://localhost/ows/34/AdServer.cfm">
```

When the browser encounters the `<IMG>` tag, it communicates with the `AdServer.cfm` template and displays whatever image content is returned. As long as you use a fully qualified URL (including the `http://`), you are free to save the HTML file on any server in the world. It doesn't have to be on the same machine as the ad server.

Handling Click-Throughs

Listing 32.9 now correctly serves up banner ads, but it can't be really considered a fully featured ad-management system at this point. For instance, what about tracking the number of ads served? What happens when the user clicks an ad?

Unfortunately, there isn't space to develop and discuss a world-class ad server here that would meet everyone's needs. That said, a few lines of code can be added to the `AdServer.cfm` template to make the example a bit more complete.

NOTE

The main purpose of these examples is to demonstrate how `<CFCONTENT>` can be combined with client or session variables to build a useful application based on serving media files, such as images. If you were building a real-world ad-server utility, you would probably do some things differently (for instance, use a database).

Listing 32.10 shows an `Application.cfm` file that turns on ColdFusion's Session Management feature and then creates a structure called `SESSION.AdsOnPages` if it doesn't exist already. Because the structure is kept in the `SESSION` scope, it is maintained separately for each user who is shown an ad. You'll see how this structure is used in a moment.

NOTE

The following examples use variables in the `SESSION` scope without locking the accesses with the `<CFLOCK>` tag. This is an acceptable practice if the Single Threaded Sessions option is checked in the Locking page of the ColdFusion Administrator. Otherwise, you must add `<CFLOCK>` tags around all accesses to the `SESSION` scope. See the section "Locking Revisited" in Chapter 17, "Working with Sessions."

Listing 32.10 `Application.cfm`—Getting Ready to Track Ad Views Per Page and Per User

```
<!---
  Filename:     Application.cfm
  Created by:   Nate Weiss (NMW)
  Please Note:  Executes for every page request!
--->

<!--- Give application a name, and enable Session variables --->
<CFAPPLICATION
  NAME="AdServer"
  SESSIONMANAGEMENT="Yes">

<!--- Make sure "AdsOnPage" structure exists for this Session --->
<!--- Each time we serve an ad, we'll record page/ad in this. --->
<CFIF NOT IsDefined("Session.AdsOnPages")>
  <CFSET SESSION.AdsOnPages = StructNew()>
</CFIF>
```

The version of `AdServer.cfm` in Listing 32.11 is almost the same as Listing 32.9. Only one line of code has been added, before the `<CFCONTENT>` tag at the end:

```
<!--- Record fact that this ad is now placed on this page --->
<!--- You could record the ad-showing in database instead --->
<CFSET SESSION.AdsOnPages[CGI.HTTP_REFERER] = AdFileName>
```

Listing 32.11 `AdServer2.cfm`—Recording Each Ad Hit in a Session Variable

```
<!---
  Filename:     AdServer2.cfm
  Created by:   Nate Weiss (NMW)
  Date Created: 2/18/2001
--->

<!--- Ad Images are in "Ads" subfolder, within current folder --->
<CFSET AdDir = GetDirectoryFromPath(GetCurrentTemplatePath()) & "Ads\">

<!--- Get a listing of GIF image files in the Ads folder --->
<!--- Result is a query object with a row for each image --->
<CFDIRECTORY
  DIRECTORY="#AdDir#"
  ACTION="LIST"
  FILTER="*.gif"
  NAME="GetAds">

<!--- Pick random number between one and number of images --->
<CFSET AdNum  = RandRange(1, GetAds.RecordCount)>
<!--- Grab the filename in chosen row of the GetAds query --->
<CFSET AdFileName = GetAds.Name[AdNum]>
<!--- Prepend dir to get full filesystem path to ad image --->
<CFSET AdFilePath = AdDir & AdFileName>

<!--- Record fact that this ad is now placed on this page --->
<!--- You could record the ad-showing in database instead --->
<CFSET SESSION.AdsOnPages[CGI.HTTP_REFERER] = AdFileName>

<!--- Send the chosen image back to the client --->
<CFCONTENT TYPE="image/gif" FILE="#AdFilePath#">
```

The idea here is to exploit the fact that the browser will provide the referring page when it requests the image from the `AdServer.cfm` page. That is, the URL of the page in which the image is to appear will be available in the `CGI.HTTP_REFERER` variable. Therefore, the `<CFSET>` line shown previously places a new name/value pair in the `AdsOnPages` structure, where the referring URL is used as the name portion of the structure entry, and the filename of the ad being shown is the value portion.

Later, you easily can look up the ad's filename using the page's URL as the structure's key. In other words, the template is now tracking which users have seen which ads, on which pages. (See Chapter 8, "Using ColdFusion," for a discussion about ColdFusion structures.)

NOTE

For more information about `HTTP_REFERER` and the other variables in the CGI scope, see Appendix D, "Special ColdFusion Variables and Result Codes."

NOTE

In addition to (or instead of) maintaining the `AdsOnPages` structure, you could record each banner ad view in a database table—for example, called `AdViews`. The table might have columns such as `AdID`, `PageURL`, and `ViewDate`. This would enable you to easily create reports based on usage.

Now that each ad view is being recorded in the `AdsOnPages` structure, you can create a template to respond when an ad is clicked. You can use the code in Listing 32.12, assuming that this is template that will be called when a user clicks an ad. It again uses the `CGI.HTTP_REFERER` variable to determine the URL of the referring document, which should be the URL that the ad is on. The idea is to use the referrer URL to look up the last ad shown to this user on that page.

Listing 32.12 `AdClick.cfm`—Responding to a Banner Ad Click

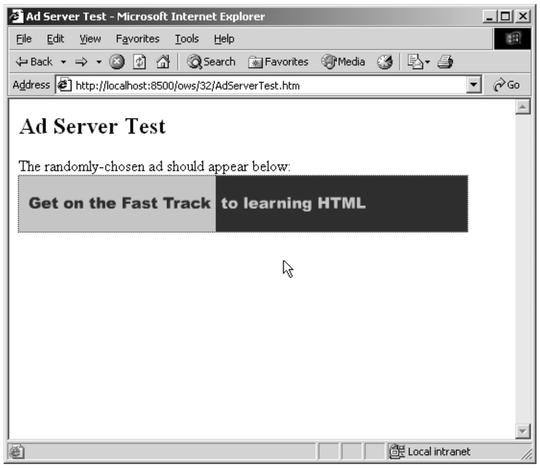
```
<!---
  Filename:     AdClick.cfm
  Created by:   Nate Weiss (NMW)
  Please Note:  This template executes when a user clicks on an ad
--->

<!--- Assuming we have shown an ad on the referring page --->
<CFIF StructKeyExists(SESSION.AdsOnPages, CGI.HTTP_REFERER)>
  <!--- What ad was last shown to this user on referring page? --->
  <CFSET AdFileName = SESSION.AdsOnPages[CGI.HTTP_REFERER]>

  <!---
    At this point, you might record the "click" in database
    and then redirect the user to the appropriate site via
    the <CFLOCATION> tag.  Here, we will simply output the
    ad that the user clicked on, just to prove the concept.
  --->
  <CFOUTPUT>
    You just came from: #CGI.HTTP_REFERER#<BR>
    You got here by clicking on the <B>#AdFileName#</B> ad.
  </CFOUTPUT>

<!--- If "KeyExists" fails, we don't know what ad was shown --->
<CFELSE>
  We don't have any ad on record for you.  Perhaps you didn't
  come here after seeing an ad, or your session timed out.
</CFIF>
```

First, the `StructKeyExists()` function is used to see whether a record exists for the URL. If so, the `<CFSET>` line retrieves the value from the structure, assigning the ad's filename to the `AdFileName` variable. This code then outputs the original filename of the ad, without redirecting the user somewhere else. As an exercise, you could take things further by using the filename to determine where to send the user in response to the ad click, log the click-through in a database, and then use `<CFLOCATION>` to redirect the user to the advertiser's Web site.

The code in Listing 32.13 can be used to test the new ad server templates. The `<IMG>` tag displays the banner ad served up by `AdServer.cfm` (Figure 32.8), and the `<A>` tag causes the `AdClick.cfm` page to be called when the ad is clicked. This is an ordinary, static HTML file that doesn't need to be processed by ColdFusion.

Figure 32.8

ColdFusion templates can serve images, such as this banner ad, instead of documents.


Ad Server Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8500/ows/32/AdServerTest.htm
Go
Local intranet

Ad Server Test

The randomly-chosen ad should appear below:

Get on the Fast Track to learning HTML

Listing 32.13 `AdServerTest.htm`—Displaying an Image Served by a ColdFusion Template

```
<!---
  Filename:     AdServerTest.htm
  Created by:   Nate Weiss (NMW)
  Please Note:  This is a static HTML file, not a CF template
--->

<HTML>
<HEAD><TITLE>Ad Server Test</TITLE></HEAD>
<BODY>

<H2>Ad Server Test</H2>

The randomly-chosen ad should appear below:<BR>

<!--- Include banner ad supplied by "Ad Server" --->
<A HREF="AdClick.cfm">
<IMG SRC="AdServer.cfm" WIDTH="468" HEIGHT="60" ALT="" BORDER="0"></A>

</BODY>
</HTML>
```

Other Uses

You could apply this same idea—using ColdFusion to serve up media files—to many other situations. Here are just a few ideas.

Creating or Changing the Ads On-the-Fly

Using third-party extensions to ColdFusion, you can create or manipulate images on-the-fly. This means you could adapt the ad server idea to create personalized images of some kind (such as greeting cards). Take a look in the ColdFusion Developers Exchange at `http://www.allaire.com/developer/gallery` to find out which CFX and other tags are available for creating dynamic images with ColdFusion.

NOTE

If you did start to create images on-the-fly for each user, you could use the `DELETEFILE="Yes"` attribute for the `<CFCONTENT>` tag. This would cause each dynamically created image to be deleted from disk after it was delivered to the browser.

Other Types of Media Content

You also could adapt this ad server idea to serve up other types of multimedia content, such as Macromedia Flash movies, or audio or video files. As long as you provide the correct MIME content type to the `TYPE` attribute of the `<CFCONTENT>` tag, you can serve up just about any type of content with ColdFusion.

Secure Downloads

If you look back at Listing 32.9, you will notice that `<CFCONTENT>`'s `FILE` attribute requires a full filesystem path (including the `c:\`, for instance, if you're using ColdFusion on a Windows machine). This means the banner images could be stored anywhere on the server's drives or local network. That is, you are free to store the images in a folder that is not within the Web server's document root, and thus not accessible via the Internet using an ordinary URL (a URL ending with `.gif`). The only way to get to the files would be via the `AdServer.cfm` template. In fact, if you think about it, there isn't even any way for the outside world to know what the name of each image file is.

This means you easily can set up a secure download system for documents or other important files your company wants to be accessible to only certain people. Or, you could set up a software download page that requires people to register (or pay) before they can obtain your software. Because ColdFusion is in charge, you can use whatever queries or other special processing you need to determine whether the user has the right to get a particular image or file.

NOTE

If you want to give the downloaded file a name, or if you find that you don't get the Save As behavior you want, try using a `Content-Disposition` header, such as the Word and Excel examples earlier in this chapter.

Getting Started with Wireless Applications

Just as you can use ColdFusion to create interactive applications for Web browsers, you can use ColdFusion to create interactive applications for cell phones and other wireless devices. This opens up exciting ways to provide information and services to your users while they are on the move.

This section explains what's needed to create wireless applications with ColdFusion, using the WAP standard. You also can adapt these basic concepts to produce wireless applications for PalmOS handhelds, I-Mode–compliant devices, or any other wireless standard that uses text-based markup.

Basic Concepts

Before you get started, you need to understand the ways in which wireless development is different from normal Web application development. You will see that the framework for building wireless applications borrows very heavily from the Web development world. That's good for you because you will be able to apply your knowledge of ColdFusion to the wireless universe very quickly.

Here are some important terms to keep in mind as you follow along:

- **WAP**—WAP stands for Wireless Application Protocol. It defines the way wireless devices talk to Web servers. Although the analogy is not perfect, you can think of WAP as the wireless equivalent to HTTP. It grew out of efforts led by a company called Phone.com (now `openwave.com`) to create a standard for wireless application development. WAP's home on the Web is the Open Mobile Alliance site, at `http://www.openmobilealliance.org`.
- **WML**—WML stands for Wireless Markup Language. WML is the wireless equivalent to HTML and is similar to HTML in many respects. WML grew out of an earlier specification called HDML (Handheld Device Markup Language).
- **WAP Device**—Any cell phone or other device that understands WAP and WML. If you don't have such a phone yourself, you can use a phone simulator on your computer while you develop your application; it looks and behaves similarly to a real cell phone.
- **WAP Gateway**—A server that sits between a wireless device and the Web. When a user uses her cell phone to connect to your ColdFusion application, the communication passes through a gateway computer, which is generally supplied by the user's cell phone company.

Installing the UP.SDK

The first thing to do is to install the UP.SDK on your workstation. The UP.SDK is a set of tools for wireless application developers. Installing it gives you access to an onscreen simulation of a cell phone, which you can use to develop and test your applications.

Before you can begin, you will need to download the UP.SDK from `http://developer.openwave.com`.

To install the UP.SDK, follow these steps:

1. Download the setup program from `http://developer.openwave.com`. Depending on the version, the filename will be similar to `upsdkW41e.exe`.
2. Double-click the setup program to start the installation process.
3. Follow the simple instructions to install the UP.SDK.

NOTE

The figures and examples in this book assume that you are using version 4.1 of the UP.SDK. If you are using a different version, the figures may look slightly different from what you see on your screen.

A Crash Course in WML

Instead of writing your ColdFusion templates to generate HTML as you would normally, you write them to generate WML.

You'll find that many tags from HTML have been adopted by WML with hardly any changes. In general, most of the differences between the languages have to do with preserving bandwidth because cell phones and other devices have relatively slow connections. Also, cell phones obviously don't have all the processing power that an average computer does, so the language has been kept very simple.

WML has a number of rules, which must be followed exactly:

- **WML is case sensitive**—In WML, tag names and attributes are case sensitive; therefore, you can't type WML tags using your choice of uppercase or lowercase as you can with HTML. You must type them as shown.
- **All tags must be nested properly**—In WML, all tags must have opening and closing tags. In HTML, you can use a <p> tag without ever providing the matching </p> tag—not so in WML.
- **Unpaired tags must use slash notation**—You are familiar with the <br> tag in HTML. The equivalent in WML is the <br/> tag. The slash indicates that no matching </br> tag exists, so the device's parser needn't waste time looking for it. Actually, this slash notation is a shortcut for writing <br></br>, which is also valid WML (but more tedious to type). The same rule applies to other unpaired tags, so you will see the slash used at the end of <img> and <a> tags in WML code.

NOTE

Most of these rules are the result of the fact that the WML language is actually a subset of XML. HTML is not directly related to XML and doesn't adhere to many of XML's rules regarding syntax.

Explaining the entire WML language in one chapter isn't possible. You will learn enough to get you started, though. Table 32.3 shows the WML tags are used in this chapter's code samples. The table also attempts to explain what each tag's equivalent is in HTML. Although the comparisons are not always perfect, they should help you feel comfortable as you start looking at the examples.

Table 32.3 Common WML Tags and Their Rough Equivalents in HTML

WML TAG	PURPOSE	HTML COUNTERPART
`<wml>`	Marks beginning and end of a WML document, more commonly called a deck	`<html>`
`<card>`	Marks beginning and end of a card	`<body>`
`<p>`	Marks beginning and end of a paragraph	`<p>`
`<a>`	Defines a hyperlink to another card	`<a>`
`<br/>`	Starts a new line	`<br>`
`<do>`	Assigns actions to buttons on the WAP device	`<button>`
`<go>`	Performs an action, such as submitting data or linking to another page	`<form>`
`<input>`	Collects input from user	`<input>`
`<postfield>`	Submits form-style input to the server	hidden `<input>`

Your First WML Card

In WML, you deal with cards instead of pages. Just as a Web browser always shows one page at a time (ignoring frames for the moment), a WAP device always shows one card at a time. The first card to write is a home card for your wireless application. This gives the user some basic links to the information you are going to provide.

Writing the Code

Listing 32.14 shows a ColdFusion template that displays a simple welcome message to the user. It also presents the user with two links that he can use to navigate to get information about Orange Whip Studio's movies.

This chapter includes several version of this template. Save the code in this listing as a file called `WapIndex.cfm`, not `WapIndex1.cfm`.

Listing 32.14 `WapIndex1.cfm`—Presenting Simple Text and Links to the User

```
<!---
  Filename:     WapIndex1.cfm
  Created by:   Nate Weiss (NMW)
  Purpose:      A home card for browsing and searching films via WAP
--->
```

Listing 32.14 (continued)

```
<!--- Send back the proper WAP content type --->
<CFCONTENT TYPE="text/vnd.wap.wml">

<!--- Send back the proper WAP "prologue" --->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
  "http://www.phone.com/dtd/wml11.dtd" >


<wml>
  <card>
    <p>
      <!--- Welcome message --->
      <b>OrangeWAP Studios</b><br/>
      <i>Celluloid by Cell</i><br/>

      <!--- Links to other pages --->
      <a href="#movies">Browse Movies</a><br/>
      <a href="#search">Search Movies</a><br/>
    </p>
  </card>
</wml>
```

NOTE

Because the WML language is case sensitive, it is important that you use lowercase letters as shown for `<wml>`, `<card>`, and other tags in this example.

The first line uses the `<CFCONTENT>` tag to specify that ColdFusion should respond with a content type of `text/vnd.wap.xml` when it sends back the code your template generates. Otherwise, the user's WAP device would receive the default content type of `text/html`, which it wouldn't know what to do with.

The next two lines (the `?xml` line and the `!DOCTYPE` line) include the WML document prologue, which must always appear before any actual WML content. It is very important that these lines appear exactly as shown here.

Next, the opening and closing `<wml>` tags indicate the beginning and end of the WML content itself. This is equivalent to the opening and closing `<html>` tags you are used to. Within the opening and closing `<card>` tags, a WML card is defined; it is conceptually similar to the `<body>` section of an HTML page.

Within the card, things start to look pretty familiar. The `<p>`, `<b>`, `<i>`, and `<a>` tags all do the same things they do in HTML. Just be careful to open and close them correctly, and use the slash notation for any unpaired tags, such as `<br/>`.

Trying It Out with UP.Simulator

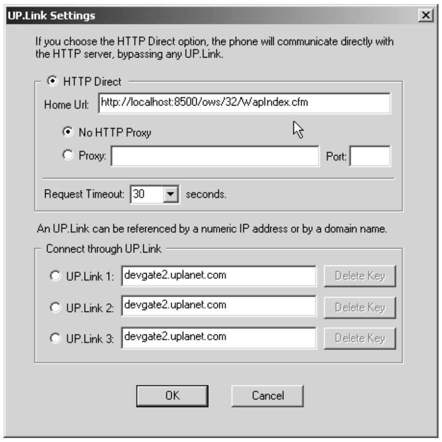
When you installed the UP.SDK, you also installed the UP.Simulator, which enables you to visit your WAP templates to see how they will look and behave with a cell phone. You can use the simulator to view the `WapIndex.cfm` page you just created.

To view your new page with the simulator, follow these steps:

1. Start the UP.Simulator by clicking its icon in the Start menu, which is in a program folder labeled `UP.SDK 4.1` or something similar—depending on the version of the UP.SDK you installed.
2. Select UP.Link Settings from the Settings menu.
3. Type the URL for the `WapIndex.cfm` template in the Home Url field (Figure 32.9). This is the same localhost URL you would use to access the template if it were a normal Web page. Leave the other settings as shown.
4. Click OK. The simulator should now connect to your Web server and display your welcome card (Figure 32.10).

Figure 32.9

Set your simulator's Home URL to your `WapIndex` template to make testing easier.



UP.Link Settings

If you choose the HTTP Direct option, the phone will communicate directly with the HTTP server, bypassing any UP.Link.

● HTTP Direct

Home Url: http://localhost:8500/ows/32/WapIndex.cfm

● No HTTP Proxy

○ Proxy: Port:

Request Timeout: 30 seconds.

An UP.Link can be referenced by a numeric IP address or by a domain name.

Connect through UP.Link

○ UP.Link 1: devgate2.uplanet.com

○ UP.Link 2: devgate2.uplanet.com

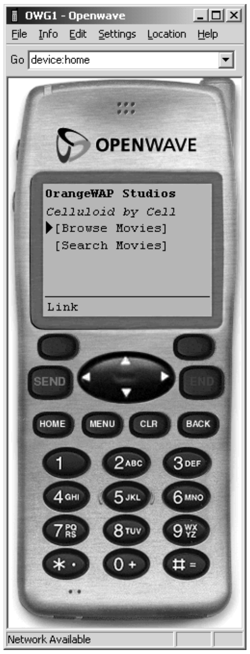
○ UP.Link 3: devgate2.uplanet.com

When you have a chance, you should explore the various menu options for the UP.Simulator to become familiar with what it can do for you. In particular, note the following:

- **Open Configuration**—You can select this from the File menu to simulate various makes and models of real-world WAP phones.
- **Reload**—You can select this from the Edit menu to reload the current card. This is similar to using the Reload button on a Web browser.
- **Source**—You can select this from the Info menu to view the source code for the current deck. This is similar to Viewing Source in a Web browser.
- **Clear Cache**—You can select this from the Edit menu to clear the simulator's cache. It also returns you to the Home URL you entered previously. It's a good way to start over after you change some code.

Figure 32.10

Your WML is interpreted by the simulated phone and displayed on its small LCD screen.



OrangeWAP Studios
Celluloid by Cell
[Browse Movies]
[Search Movies]
Link

Using `<CFINCLUDE>` for the Prologue

As explained previously, it is very important to get the `<CFCONTENT>` and prologue parts of Listing 32.14 exactly right in every WML template you build. For that reason, it would probably be helpful to put those lines in a separate template, so you can just use `<CFINCLUDE>` to include the proper prologue each time.

Listing 32.15 shows a template called `WapIncludePrologue.cfm`. Listing 32.16 is the `WapIndex.cfm` file again, this time using `<CFINCLUDE>` to include the content type and prologue—much easier.

Listing 32.15 `WapIncludePrologue.cfm`—Sending the WAP Prologue in an Included Template

```
<!---
  Filename:     WAPIncludePrologue.cfm
  Created by:   Nate Weiss (NMW)
  Please Note:  Can be used via <CFINCLUDE>, or as a CFML
                Custom Tag (<CF_WapIncludePrologue>)
--->

<!--- Send back the proper WAP content type --->
<CFCONTENT TYPE="text/vnd.wap.wml">

<!--- Send back the proper WAP "prologue" --->
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
  "http://www.phone.com/dtd/wml11.dtd" >
```

Listing 32.16 `WapIndex2.cfm`—Using `<CFINCLUDE>` to Include WAP Content Type and Prologue

```
<!---
  Filename:     WapIndex2.cfm
  Created by:   Nate Weiss (NMW)
  Purpose:      A home card for browsing and searching films via WAP
--->

<!--- Include WAP Content-Type and Prologue --->
<CFINCLUDE TEMPLATE="WapIncludePrologue.cfm">

<wml>

  <card>
    <p>
      <!--- Welcome message --->
      <b>OrangeWAP Studios</b><br/>
      <i>Celluloid by Cell</i><br/>

      <!--- Links to other pages --->
      <a href="#movies">Browse Movies</a><br/>
      <a href="#search">Search Movies</a><br/>
    </p>
  </card>

</wml>
```

NOTE

Listing 32.15 can also be used as a CFML Custom tag. Just save `WapIncludePrologue.cfm` in ColdFusion's `CustomTags` folder. You can then replace the `<CFINCLUDE>` line in Listing 32.16 with `<CF_WapIncludePrologue>`. See Chapter 20, "Building Reusable Components," for details.

Multiple Cards in One Deck

One interesting feature of WML is that you can include more than one card within each document. That is, each `<wml>` tag can include more than one `<card>` tag. The WML document, or collection of cards, is therefore called a Deck. You can include links from one card to another, and the user can move from card to card within the deck without recontacting the server.

NOTE

You certainly don't have to include multiple cards in each of your WML templates, but it is definitely encouraged. Because WAP devices generally have slow connections, you should keep the number of requests to your server to a minimum. By including multiple cards with each request, you potentially save a lot of time for your users.

The version of `WapIndex.cfm` in Listing 32.17 creates a simple deck of three cards. From the first card, the user can navigate to the other two cards. Note that each card has been given an `id` attribute. Links to other cards in the same deck are made by referring to the target card's `id`, using the `#` notation shown in the two `href` attributes.

Listing 32.17 `WapIndex3.cfm`—Including Multiple Cards in the Same Deck

```
<!---
  Filename:     WapIndex3.cfm
  Created by:   Nate Weiss (NMW)
  Purpose:      A home card for browsing and searching films via WAP
--->

<!--- Include WAP Content-Type and Prologue --->
<CFINCLUDE TEMPLATE="WapIncludePrologue.cfm">


<wml>
  <card id="home">
    <p>
      <!--- Welcome message --->
      <b>OrangeWAP Studios</b><br/>
      <i>Celluloid by Cell</i><br/>

      <!--- Links to other pages --->
      <a href="#movies">Browse Movies</a><br/>
      <a href="#search">Search Movies</a><br/>
    </p>
  </card>

  <card id="movies">
    <p>
      <b>Movie List</b>
    </p>
  </card>

  <card id="search">
    <p>
      <b>Movie Search</b>
    </p>
  </card>
</wml>
```

NOTE

Much of the finesse of WAP development is sending a sensible set of cards in each request. Even if you don't know that the user will want to look at all the cards in a deck, you still should include them if you can. Of course, you don't want to include cards that no one ever seems to use, either. As long as there is a reasonable chance the user will navigate to each card, it's worth putting the card in there. The additional cost (in time) of including each additional card is generally much lower than the cost of additional connections back to your server.

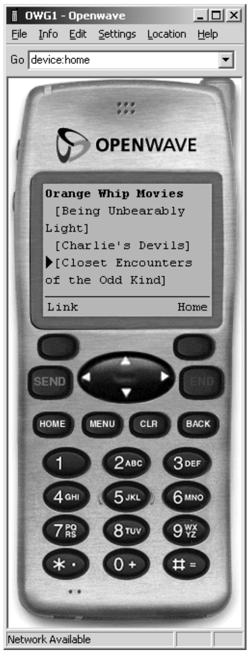
Creating Data-Driven Cards

So far, you have created static cards that always look the same. What about creating cards dynamically—for instance to display information from a database? It's really quite simple. Just use `<CFQUERY>` and `<CFOUTPUT>` tags as you would normally, ensuring that the content your template generates is valid WML.

Listing 32.18 demonstrates how easy generating data-driven WML is. This is essentially the same as the last listing, except for the addition of a query to retrieve a list of current movies. Then, within the movies card, a link in the form of the <a> tag is generated for each film (Figure 32.11).

Figure 32.11

Creating data-driven wireless applications is easy with ColdFusion and WML.



```
Orange Whip Movies
 [Being Unbearably
Light]
 [Charlie's Devils]
▶[Closet Encounters
of the Odd Kind]
Link            Home
```

Listing 32.18 `WapIndex4.cfm`—Generating WML Dynamically from Query Results

```
<!---
  Filename:     WapIndex4.cfm
  Created by:   Nate Weiss (NMW)
  Purpose:      A home card for browsing and searching films via WAP
--->

<!--- Include WAP Content-Type and Prologue --->
<CFINCLUDE TEMPLATE="WapIncludePrologue.cfm">

<!--- Get movies from database --->
<CFQUERY
  DATASOURCE="OrangeWhipStudios"
  NAME="GetMovies"
  CACHEDWITHIN="#CreateTimeSpan(0,0,15,0)#">
  SELECT FilmID, MovieTitle, Summary
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>
```

Listing 32.18 (continued)

```
<wml>

  <card id="home">
    <p>
      <!--- Welcome message --->
      <b>OrangeWAP Studios</b><br/>
      <i>Celluloid by Cell</i><br/>

      <!--- Links to other pages --->
      <a href="#movies">Browse Movies</a><br/>
      <a href="#search">Search Movies</a><br/>
    </p>
  </card>

  <card id="movies">
    <!--- Present a link for each movie --->
    <p>
      <b>Orange Whip Movies</b>
      <CFOUTPUT QUERY="GetMovies">
        <CFSET MovieCardURL = "WapMov.cfm?FilmID=#FilmID#">
        <a href="#MovieCardURL#">#MovieTitle#</a><br/>
      </CFOUTPUT>
    </p>
  </card>


  <card id="search">
    <p>
      <b>Movie Search</b>
    </p>
  </card>
</wml>
```

Browsing Through Records

When the user clicks any of the links in Listing 32.18, she will be linked to `WapMov.cfm`, with the selected film's ID passed as a URL parameter. Listing 32.19 shows one way that template can be written.

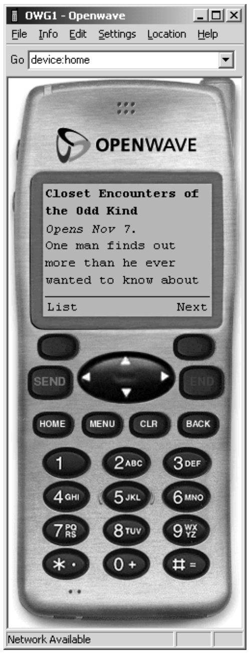
Rather than responding with a single card, this code sends back a card for the requested movie, plus one card each for the next four movies. Each card enables the user to click the Next button to navigate to the next card in the deck (Figure 32.12). When the user clicks the Next button on the fifth card, the template is revisited: As the user scrolls through the list of films, the server is contacted only for every fifth film.

NOTE

Most WAP devices display an error message if your code contains more than 2,000 characters or so. You can adjust the number of cards sent back per deck by adjusting the `MaxRowsPerDeck` variable. In general, the number should be as high as possible without the whole deck exceeding the 2,000-character limit. See the UP.SDK documentation for more details regarding code-size limits.

Figure 32.12

The user can browse through cards using the Next link.



Closet Encounters of the Odd Kind
Opens Nov 7.
One man finds out more than he ever wanted to know about

List Next

Listing 32.19 `WapMov.cfm`—Enabling the User to Browse Through the List of Movies

```
<!---
  Filename:     WapMov.cfm
  Created by:   Nate Weiss (NMW)
  Purpose:      Displays film information for WAP users
--->

<!--- Include WAP Content-Type and Prologue --->
<CFINCLUDE TEMPLATE="WapIncludePrologue.cfm">

<!--- We need to limit how many cards we send --->
<CFSET MaxCardsPerDeck = 5>

<!--- Get movies from database --->
<CFQUERY
  DATASOURCE="ows"
  NAME="GetMovies"
  CACHEDWITHIN="#CreateTimeSpan(0,0,15,0)#">
  SELECT FilmID, MovieTitle, Summary, DateInTheaters
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- A FilmID must be passed in the URL --->
<CFPARAM NAME="URL.FilmID" TYPE="numeric">
<!--- Find the passed FilmID's row in query --->
<CFSET StartRow = ListFind(ValueList(GetMovies.FilmID), URL.FilmID)>
<!--- We'll start at that row, and end 5 rows later --->
```

Listing 32.19 (continued)

```
<CFSET EndRow   = StartRow + MaxCardsPerDeck - 1>




<wml>
  <!--- All cards need a way back to movie list --->
  <template>
    <do type="accept" label="List">
      <go href="WapIndex.cfm#movies"/>
    </do>
  </template>

  <!--- Create a card for each movie --->
  <CFOUTPUT QUERY="GetMovies"
    STARTROW="#StartRow#"
    MAXROWS="#MaxCardsPerDeck#">

    <card id="Film#FilmID#">

      <!--- Show "Next" navigation, unless at end --->
      <CFIF CurrentRow LT RecordCount>
        <!--- What is the next film? --->
        <CFSET NextID = FilmID[CurrentRow + 1]>

        <do type="options" label="Next">
          <!--- If next film is in this deck --->
          <CFIF CurrentRow NEQ EndRow>
            <go href="##Film#NextID#"/>
          <!--- If next film not in this deck --->
          <CFELSE>
            <go href="?FilmID=#NextID#"/>
          </CFIF>
        </do>
      </CFIF>

      <!--- Display information about this movie--->
      <p>
        <b>#MovieTitle#</b><br/>
        <i>Opens #DateFormat(DateInTheaters, "mmm d")#.</i><br/>
        #Summary#<br/>
      </p>
    </card>
  </CFOUTPUT>
</wml>
```

Responding to Form Input

Just like normal Web applications, your WAP applications can include forms for the user to fill in and submit. For instance, you might want the user to be able to search for a movie by typing in a word or two.

The last version of `WapIndex.cfm` included a movie search card, but it didn't actually work. To enable the user to specify his search criteria, the `search` card from Listing 32.18 needs to be completed, as shown in Listing 32.20.

Listing 32.20 `WapIndex5.cfm`—Allowing the User to Run Searches

```
<!---
  Filename:     WapIndex5.cfm
  Created by:   Nate Weiss (NMW)
  Purpose:      A home card for browsing and searching films via WAP
--->

<!--- Include WAP Content-Type and Prologue --->
<CFINCLUDE TEMPLATE="WapIncludePrologue.cfm">

<!--- Get movies from database --->
<CFQUERY
  DATASOURCE="ows"
  NAME="GetMovies"
  CACHEDWITHIN="#CreateTimeSpan(0,0,15,0)#">
  SELECT FilmID, MovieTitle, Summary
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>


<wml>

  <card id="home">
    <p>
      <!--- Welcome message --->
      <b>OrangeWAP Studios</b><br/>
      <i>Celluloid by Cell</i><br/>

      <!--- Links to other pages --->
      <a href="#movies">Browse Movies</a><br/>
      <a href="#search">Search Movies</a><br/>
    </p>
  </card>

  <card id="movies">
    <!--- Give user a way to get back home --->
    <do type="options" label="Home">
      <go href="#home"/>
    </do>

    <!--- Present a link for each movie --->
    <p>
      <b>Orange Whip Movies</b>
      <CFOUTPUT QUERY="GetMovies">
        <a href="WapMov.cfm?FilmID=#FilmID#">#MovieTitle#</a><br/>
      </CFOUTPUT>
    </p>
  </card>


  <card id="search">
    <!--- Run search when "Accept" button pressed --->
    <do type="accept" label="Submit">
      <go method="post" href="WapSearch.cfm?">
        <postfield name="criteria" value="$criteria"/>
      </go>
    </do>
```

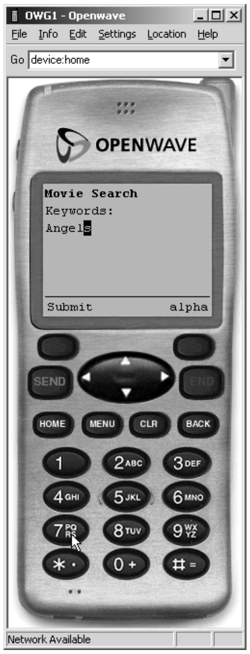
Listing 32.20 (CONTINUED)

```
        <!--- Allow user to type search criteria --->
        <p>
          <b>Movie Search</b><br/>
          Keywords: <input name="criteria"/>
        </p>
      </card>
    </wml>
```

This is roughly equivalent to a simple search form on an HTML page. The `<input>` field gives the user a place to enter his search criteria, and the `<do>` tag gives the user a way to submit the search by clicking the Accept button on the phone (see Figure 32.13).

Figure 32.13

The user can key in films to search for with the numeric buttons on the phone.



Movie Search
Keywords:
Angels

When the user clicks the Accept button, the `<go>` action fires, essentially acting like a `<form>` tag in HTML. The `method` attribute can be set to `get` or `post`; the `href` attribute, on the other hand, indicates the page to submit to, such as a Web form's `action` attribute. And just as with Web forms, you should use `method="post"`. The `<postfield>` tag causes the value of the `<input>` to be submitted as a form-like value.

NOTE

The value of `<input>` automatically becomes known to the phone as a variable called `$criteria`, which can be used in various ways. Here, the variable is just passed to the `<postfield>` tag so its value gets submitted to ColdFusion. But you can do a lot more with these variables. Refer to the documentation that was installed with the UP.SDK to see what else you can do with WAP variables and the `$` notation used here.

Now you can write a ColdFusion template called `WapSearch.cfm`. Listing 32.21 shows one way to respond to the user's search request. Note that you can refer to the user's search criteria as `#Form.Criteria#`, just as if the search request were coming from a normal Web form.

Listing 32.21 `WapSearch.cfm`—Responding to Form Input

```
<!---
  Filename:     WapSearch.cfm
  Created by:   Nate Weiss (NMW)
  Purpose:      Provides a simple film search interface for WAP users
--->

<!--- Include WAP Content-Type and Prologue --->
<CFINCLUDE TEMPLATE="WapIncludePrologue.cfm">

<!--- We must have search criteria --->
<CFPARAM NAME="Form.Criteria" TYPE="string">

<!--- Get movies from database --->
<CFQUERY
  DATASOURCE="ows"
  NAME="GetMovies"
  CACHEDWITHIN="#CreateTimeSpan(0,0,15,0)#">
  SELECT FilmID, MovieTitle, Summary
  FROM Films
  WHERE MovieTitle LIKE '%#Form.Criteria#%'
     OR PitchText  LIKE '%#Form.Criteria#%'
     OR Summary    LIKE '%#Form.Criteria#%'
  ORDER BY MovieTitle
</CFQUERY>


<wml>
  <card id="search">
    <!--- Give user way back to "Search" card --->
    <do type="options" label="Again">
      <go href="WapIndex.cfm#search"/>
    </do>

    <p>
      <!--- Show "Results" message, with criteria --->
      <b>Search Results For</b><br/>
      <CFOUTPUT>&quot;<b>#Form.Criteria#</b>&quot;<br/></CFOUTPUT>

      <!--- For each matching movie, provide link --->
      <CFOUTPUT QUERY="GetMovies">
        <a href="WapMov.cfm?FilmID=#FilmID#">#MovieTitle#</a><br/>
      </CFOUTPUT>
    </p>
  </card>
</wml>
```

NOTE

It's not easy for users to type on the tiny keypads cell phones come with. Try to keep the amount of typing down to a bare minimum. Elaborate data-entry screens with many fields probably will be too time-consuming for your users to actually use.

Learning More

Even though these examples should give you enough to hit the ground running, this section has really only scratched the surface with regard to what you can do with WML. You probably need to learn more about the WML language before you can build the wireless application of your dreams. All the following are great resources:

- The WAP, WML, and WMLScript documentation that was installed when you installed the UP.SDK
- The WAP Forum Web site, at `http://www.openmobilealliance.org`
- The Openwave Web site, at `http://www.openwave.com`
- *WAP Development with WML and WMLScript*, by Ben Forta
- *Learning WML & WMLScript*, by Martin Frost (O'Reilly)