CHAPTER 23

IMPROVING THE USER EXPERIENCE

In this chapter

Usability Considerations

Remembering Settings

Creating Next N Records Interfaces

Returning Page Output Right Away with CFFLUSH

USABILITY CONSIDERATIONS

This chapter concentrates on issues regarding the overall user experience and various ways it can be improved. The phrase *user experience* is purposefully a bit vague and is hard to measure quantitatively. Different people will define it differently. For the purposes of this discussion, think of the quality of the user experience as being affected mainly by the combination of application performance, usability, and friendliness.

In other words, do users have a pleasant experience when they use your application?

PUTTING YOURSELF IN THE USER'S SHOES

One of the best ways to ensure that your application is pleasant for your users to use is to simply keep them in mind as you do your development work. When you are deep into a development project, perhaps rushing to meet a deadline, it's easy to just produce code that works well enough for you to move on to the next task, without asking yourself whether it's really good enough for the user.

So, even if you are not responsible for the design or navigation, you still should keep the user in mind as you put together each data-entry screen or code each query. If users are happy, your application will probably be successful. If they spend too much time waiting or get confused, your application will probably not be successful. In most situations, especially applications aimed at the general public, it's as simple as that.

THINKING ABOUT NAVIGATION

Entire books have been written on great ways to set up navigation elements for Web sites. Navigation elements should not only look good, but also be clear and easy to use. There shouldn't be too many choices, especially on the first page of your application. At the same time, you will notice that most Web sites try to ensure that the most important content is no more than three levels (clicks) deep into the navigation structure. Most usability scholars agree that the most important or most commonly used items should appear before the less important items, even at the cost of the order being somewhat less predictable.

Studying the navigation elements used by your favorite Web sites—the ones you use often—can be helpful. What do these sites have in common? What is the theory behind the navigation on each page? For instance, does it adapt itself to context, or does it remain exactly consistent from page to page? Why are some things on the home page but other things on a second-level page? Try to come up with rules that explain which items appear on which pages, and where. For instance, some sites tend to put verbs in a toolbar at the top of each page and nouns in the left margin. Other sites might do the reverse. Try to come up with similar rules for what goes where within your own application.

Tip

Good discussions on navigation and other design elements can be found on a number of developer-related Web sites. One good place to start is the Dimitry's Design Lab section at the WebReference Web site (`http://www.webreference.com/dlab`).

WHY IT'S GOOD TO BE PREDICTABLE

When describing a friend or a spouse, the word *predictable* doesn't sound like much of a compliment. But when describing a Web application, predictability is almost always something that should be actively pursued and cherished. As users move from page to page, they will feel most comfortable if they can predict, even intuit, what is going to appear next.

ANTICIPATING THE USER'S NEXT MOVE

As you are putting together a page, don't just think about what the user is going to do on that page. Try to figure out what the user is likely to do *next*. If the user is filling out a registration form, he might want to know what your company's privacy policy is. Or, if the user is reading a press release, he might appreciate links to the company's corporate information and information about its management team.

In general, on any page, try to put yourself in the user's shoes and ask yourself whether it's clear how to get to the next step or the next piece of information.

SCRIPTING, ROLLOVERS, AND WIDGETS

JavaScript and Dynamic HTML can go a long way toward making your applications feel more exciting to your users. They also can cause problems of their own. For instance, image rollovers are great when used judiciously but can really slow a page down if too many rollovers are used.

In particular, Dynamic HTML functionality is notorious for behaving differently from browser to browser, version to version, and platform to platform. If you are going to use Dynamic HTML, try to find cross-browser scripts you can adapt until you come to understand what all the specific limitations are. As a start, `www.webreference.com`, `www.builder.com`, and `www.dhtmlzone.com` are good places to look for scripts that work reasonably well in various browsers.

DEALING WITH PROBLEMS GRACEFULLY

Hopefully, your application will never encounter any serious problems or error conditions. Unfortunately, as great as ColdFusion is, your application is bound to run into some type of problem at some point. Try to ensure that any error messages still seem friendly and encouraging to the user and don't cause her to lose her trust in your Web site.

For instance, consider customizing all error messages so that they match the look and feel of your site. For instructions, see the section "Customizing the Look of Error Messages" in Chapter 19, "Introducing the Web Application Framework." See also Chapter 33, "Error Handling."

EASING THE BROWSER'S BURDEN

If an application isn't running as fast as you want it to, you usually should try to look for a source of trouble on the server side, such a `<CFQUERY>` tag that is taking longer to execute than it should. However, you also should spend some time thinking about how much work the browser machine is doing to display your pages.

DEALING WITH IMAGE SIZE

No matter how ardently you work to make your application generate sensible HTML, and no matter how hard you work to ensure all your queries and other server-side code runs quickly, it is all too easy to slow down your pages with a lot of large images. Not only can large images take a long time to download, they also take up room in the browser machine's memory, which can have an effect on the user's computer (depending on how much RAM and virtual memory the user has available).

Here are some suggestions to keep image size in check:

- **Create or resave your images using a program that knows how to compress or optimize images for use on the Web**—In general, you give up a bit of image quality for a smaller file size; the trade-off is generally worth it. For instance, recent versions of Adobe Photoshop include the terrific Save for Web option on the File menu, which enables you to preview how your images will look after they are optimized. Macromedia's Fireworks product also does a great job at compressing image files and generally optimizing them for display on the Web. Other tools, some of them free or shareware, can provide similar results. One place to look for such programs is `www.shareware.com`.
- **Create or resave any JPEG images using a progressive JPEG option, so the images can be displayed in greater and greater detail as they are downloaded**—This is more pleasant for the user because she doesn't have to wait for the whole image to download before she can get a sense of the image. In Photoshop, this is also available in the Save for Web dialog box (see the previous bullet).
- **The `WIDTH` and `HEIGHT` attributes you supply to an `<IMG>` tag don't have to reflect the actual width and height of the image file**—You could, for instance, create an image file that is 50×50 pixels, yet provide `WIDTH` and `HEIGHT` attributes of `100` each. The image's file size would be much smaller, but it would take up the same amount of space on the page. Of course, it would appear pixilated, but depending on the nature of the artwork, that might be fine.
- **If you don't know an image's `WIDTH` and `HEIGHT`, you can determine the width and height by reading the dimensions dynamically using a custom tag or CFX tag**—This occurs in situations such as when you are dealing with images that have been uploaded from users (see Chapter 35, "Interacting with the Operating System"). A number of such tags are available from the ColdFusion Developers Exchange Web site. See Chapter 22, "Building Reusable Components," for details about the Developers Exchange.

- **Always try to provide a sensible ALT attribute for each <IMG> tag to describe what the image is about**—Most browsers display any text you provide in an ALT tag while the image is loading or as a tooltip when the user hovers her mouse over the image. This enables the user to anticipate what each image is going to be before it is actually displayed.
- **Consider using the LOWSRC attribute for your larger <IMG> tags**—The LOWSRC attribute enables a smaller version of the image to be displayed while the full-size image is being downloaded. At the time of this writing, no versions of Internet Explorer support LOWSRC, but it will still be of benefit to Netscape users. Consult the HTML Reference section of ColdFusion Studio's online help for details.

And, of course, you must think about whether your larger images are really necessary. Can they be eliminated, or at least be made a bit smaller? Look at some of your favorite Web sites—the ones you actually use on a daily basis. How many images do you see? Most likely, not that many. Most popular Web sites use other techniques to give visual impact to parts of a page (especially type size and background colors) and use images quite sparingly.

Tip

In any case, always provide WIDTH and HEIGHT attributes for each <IMG> tag. This enables the browser to display the rest of the page correctly before the images have been loaded. Without WIDTH and HEIGHT, the browser might have to wait until all the images have loaded to display anything, or it might have to reflow the document several times as each image is loaded. (The exact behavior in the absence of WIDTH and HEIGHT varies from browser to browser.)

USING TABLES WISELY

HTML tables are a great way to display information in any type of rows-and-columns format, such as the Next N examples shown later in this chapter. Here are a few tips to help you make the most of them, without placing an undue burden on the browser:

- **Whenever possible, provide WIDTH attributes for each <TH> or <TD> cell in the table**—This usually speeds up the display of the table. The exact behavior, however, varies somewhat from browser to browser.
- **You can specify the WIDTH attribute for an entire table as a percentage**—For instance, WIDTH="100%" tells the browser to make the table take up the entire available width of the page. You can also use percentages for the widths of each <TH> and <TD> cell. For instance, if three <TD> cells were in a table row, you could use WIDTH="50%" for the first one and WIDTH="25%" for the second two. This is helpful when you want content to spread itself evenly across the page, regardless of screen resolution. If the user resizes the page, the table automatically resizes itself as well.
- **You should try not to control an entire page's layout using tables**—This is because tables usually cannot be displayed incrementally, which means the table often will not be displayed until the closing </TABLE> tag is encountered. So, if your whole page is laid out using a single, large table, the page might not be displayed at all until the

whole page has been received (regardless of what you do with the <CFFLUSH> tag discussed later in this chapter). Sometimes, however, a table can be displayed incrementally. The exact behavior varies from browser to browser. See the COLS attribute for the <TABLE> tag in an HTML reference for details.

USING FRAMES WISELY

Frames are a nice way to separate sections of a page. A frames-based page usually takes a bit longer to appear at first because each frame must be fetched by the browser by submitting a separate page request to the server. However, after the frameset is loaded, subsequent page requests can be pretty quick (assuming that only one frame needs to be replaced, rather than the whole page).

However, some users find frames confusing—especially if a lot of them are used on a page, each with its own scrollbar. If you choose to use frames in your application, you should try to ensure that the layout is such that the content of each frame does not need to scroll.

Note

If you are using session variables, the introduction of frames makes locking all accesses to the SESSION scope especially important to prevent memory corruption. This can be done for you automatically using the ColdFusion Administrator. See Chapter 20, "Working with Sessions," for details.

USING EXTERNAL SCRIPT AND STYLE FILES

The use of JavaScript and Cascading Style Sheets (CSS) is not discussed specifically in this book, but they often become important parts of ColdFusion applications. CSS and JavaScript code usually are included as part of the HTML document itself, generally in the <HEAD> section. If you will use the same JavaScript functions or CSS classes over and over again, on a number of pages, you should consider moving the script or CSS code into separate files.

This way, the browser must download the file only once, at the beginning of each session (depending on how the browser's caching preferences have been set up by the user), rather than as a part of each page. This can make your pages display more quickly, especially for modem users—and especially if your script or CSS code is rather long.

To move frequently used JavaScript functions into a separate file, just save the JavaScript code to a file with a .js extension. The file should not include opening and closing <SCRIPT> tags; it should contain only the JavaScript code itself. Next, in place of the original <SCRIPT> block, include a reference to the .js file using the SRC attribute of the <SCRIPT> tag, like this (the closing <SCRIPT> tag is required):

```
<SCRIPT LANGUAGE="JavaScript" SRC="MyScripts.js"></SCRIPT>
```

Similarly, to move frequently used CSS code into a separate file, save the CSS code to a file with a .css extension. The file should not include any <STYLE> tags—just the CSS code

itself. Now, in place of the original <STYLE> block, include a reference to the .css file using the <LINK> tag:

```
<LINK REL="stylesheet" TYPE="text/css" HREF="MyStyles.css">
```

BROWSER COMPATIBILITY ISSUES

Not all browsers support all HTML tags. For instance, Internet Explorer does not support the <LAYER> and <SPACER> tags that are supported by Netscape Communicator, and Netscape browsers don't support the <MARQUEE> tag provided by IE. Support for various tags also differs from browser version to browser version—for instance, the <IFRAME> tag was not supported in Netscape browsers until version 6.0. Support for more advanced technologies, such as JavaScript and Dynamic HTML, differ even more widely from browser to browser.

You can use the automatic CGI.HTTP_USER_AGENT variable to determine which browser is being used to access the currently executing template. The HTTP_USER_AGENT value is a string provided by the browser for identification purposes. You can look at the string using ColdFusion's string functions to determine the browser and version number. For instance, the following line of code can be placed in Application.cfm to determine whether the user is using a Microsoft Internet Explorer browser:

```
<CFSET REQUEST.IsIE = HTTP_USER_AGENT contains "MSIE">
```

You then could use the REQUEST.IsIE variable in any of your application's pages (including custom tags or modules) to display Internet Explorer–specific content when appropriate:

```
<CFIF REQUEST.IsIE>
  <!--- Internet Explorer content goes here --->
<CFELSE>
  <!--- Non-IE content goes here --->
</CFIF>
```

Note

If you find that you need to perform more sophisticated tests about the user's environment, you might consider using a tool such as BrowserHawk from cyScape. BrowserHawk gives you a clean, managed way to determine what the user's browser is (version number, platform) and what its capabilities are (frames, DHTML, scripting, installed plug-ins, screen resolution, and so on). For your convenience, an evaluation version of BrowserHawk has been included on this book's CD-ROM.

REMEMBERING SETTINGS

One way to make your application more usable and helpful for your users is to remember certain settings or actions as they interact with it. For instance, in Chapter 20, ColdFusion's Client Management feature was used to remember the words the user last searched for. When a user returns to the search page later, he finds that his last search phrase is already filled in for him. This can improve the user experience both by saving the user time and by making him feel at home.

REMEMBERING USERNAMES AND PASSWORDS

If your application requires users to log in by providing a username and password, you might consider adding some type of "remember me" option on the login form. This would cause the username to be prefilled for the user when he next needs to log in. You can use the same basic technique used in the `SearchForm1.cfm` and `SearchForm2.cfm` templates from Chapter 20. Instead of using the `CLIENT` scope to remember the last search phrase, use it to remember and prefill the username.

Note

Of course, this makes your application less secure because anyone with physical access to the user's machine could see the user's username.

OTHER HELPFUL SETTINGS TO REMEMBER

Many other things can be remembered between user visits to save the user time and make him feel more at home:

- If the user has indicated which country he lives in, you could show him content relevant to his country each time he returns to your site, perhaps even translated into the appropriate language.
- If the user has a favorite color, you could store the color in the `CLIENT` scope and use it to set the `BGCOLOR` for the `<BODY>` tags at the top of each page.

AVOIDING THE BIG BROTHER EFFECT

After you start thinking about remembering settings for the user, it becomes clear that you probably could retain just about everything each user does, on which pages, and when. However, if your application begins to flaunt its knowledge of the user's actions in a way that is perceived as excessive, you might start to lose the trust of your users. People like it when sites are personalized for them, but no one likes to feel as if their every move is being watched and recorded. Also, be sure you aren't violating any type of privacy statement your company or client has made publicly available.

CREATING NEXT N RECORDS INTERFACES

Sooner or later, you probably will run into a situation in which you need to build what we call a *Next N interface*. A Next N interface is any Web page that enables the user to view a large number of records—say, 10 or 20 at a time. You probably have seen such interfaces yourself on a number of Web sites. They are very common on search engine Web sites, which might have 1,000 records to look through. Instead of showing you all 1,000 records at once, buttons or links labeled Next and Back move you through the records in more reasonable chunks.

ADVANTAGES OF NEXT N INTERFACES

This type of interface has a number of advantages:

- **Familiarity**—Because Next N interfaces are so common, many users expect them whenever they are presented with a large number of records. If they see a ton of records without such an interface, your application might seem unfinished.
- **Performance**—As discussed earlier, good performance is part of providing a good user experience. Because Next N interfaces put an upper boundary on the size of the generated HTML, pages that use them usually are easier on both the browser machine and the ColdFusion server.
- **Readability**—Most importantly, Next N interfaces usually enable the user to more easily find the information she is looking for, simply because reading a small page is faster than reading a large one.

WHEN TO CREATE A NEXT N INTERFACE

It will usually be obvious when you need to add a Next N interface to a particular display page. Other times, though, it is not so obvious and only becomes evident over time, as the number of records in the database grows. A year after an application is deployed, what seemed to be a nice, compact data-display page when you wrote it could get to the point where it is slow and unmanageable. So, you should consider creating some variation on the Next N interface presented in this chapter whenever you think the user might sometime need to look at a large number of records.

Tip

You might come up with an internal user interface policy stipulating that whenever a user will be presented with more than 50 records, a Next N interface should be implemented. You might pick a larger cutoff point if your users have fast connection speeds, or choose a smaller cutoff if many users will connect via slow modems.

CREATING THE BASIC INTERFACE

Say you have been asked to create a simple expense report area for Orange Whip Studio's intranet. The only instruction you have been given is to create a page in which employees can review all expenses. After talking with a few of the employees in the accounting department, you learn that they are usually most interested in viewing the most recent expenses. You decide to display the expense records in reverse order (the most recent expense first), with a Next 10 interface. This way, users can see the new expenses right away and page through the older records 10 at a time.

This section presents four versions of a typical Next N interface, with each version getting a bit more sophisticated.

LIMITING THE NUMBER OF RECORDS SHOWN

A number of approaches can be taken to create a Next N interface. Listing 23.1 demonstrates a simple, effective technique that easily can be adapted to suit your needs.

The code relies on a URL parameter named `StartRow`, which tells the template which records to display. The first time the page is displayed, `StartRow` defaults to `1`, which causes rows 1–10 to be displayed. When the user clicks the Next button, `StartRow` is passed as `11`, so rows 11–20 are displayed. The user can continue to click Next (or Back) to move through all the records.

Note

Before this listing will work, the `REQUEST.DataSource` variable needs to be set in your `Application.cfm` file, as shown in Listing 23.2.

Note

Listing 23.3, later in this chapter, also must be in place before this listing will work.

LISTING 23.1 `NextN1.cfm`—A SIMPLE NEXT N INTERFACE

```
<!--- Retrieve expense records from database --->
<CFQUERY NAME="GetExp" DATASOURCE="#REQUEST.DataSource#">
  SELECT
    f.FilmID, f.MovieTitle,
    e.Description, e.ExpenseAmount, e.ExpenseDate
  FROM
    Expenses e INNER JOIN Films f
    ON e.FilmID = f.FilmID
  ORDER BY
    e.ExpenseDate DESC
</CFQUERY>


<!--- Number of rows to display per Next/Back page  --->
<CFSET RowsPerPage = 10>
<!--- What row to start at? Assume first by default --->
<CFPARAM NAME="URL.StartRow" DEFAULT="1" TYPE="numeric">

<!--- We know the total number of rows from query   --->
<CFSET TotalRows = GetExp.RecordCount>
<!--- Last row is 10 rows past the starting row, or --->
<!--- total number of query rows, whichever is less --->
<CFSET EndRow = Min(URL.StartRow + RowsPerPage - 1, TotalRows)>
<!--- Next button goes to 1 past current end row  --->
<CFSET StartRowNext = EndRow + 1>
<!---"Back button goes back N rows from start row --->
<CFSET StartRowBack = URL.StartRow - RowsPerPage>


<!--- Page Title --->
<HTML>
<HEAD><TITLE>Expense Browser</TITLE></HEAD>
```

```
<BODY>
<CFOUTPUT><H2>#REQUEST.CompanyName# Expense Report</H2></CFOUTPUT>


<TABLE WIDTH="600" BORDER="0" CELLSPACING="0" CELLPADDING="1" COLS="3">
  <!--- Row at top of table, above column headers --->
  <TR>
    <TD COLSPAN="2">
      <!--- Message about which rows are being displayed --->
      <CFOUTPUT>
        Displaying <B>#URL.StartRow#</B> to <B>#EndRow#</B>
        of <B>#TotalRows#</B> Records<BR>
      </CFOUTPUT>
    </TD>
    <TD></TD>
    <TD ALIGN="right">
      <!--- Provide Next/Back links --->
      <CFINCLUDE TEMPLATE="NextNIncludeBackNext.cfm">
    </TD>
  </TR>

  <!--- Row for column headers --->
  <TR>
    <TH WIDTH="100">Date</TH>
    <TH WIDTH="250">Film</TH>
    <TH WIDTH="150">Expense</TH>
    <TH WIDTH="100">Amount</TH>
  </TR>

  <!--- For each query row that should be shown now --->
  <CFLOOP QUERY="GetExp" StartRow="#URL.StartRow#" ENDROW="#EndRow#">
    <CFOUTPUT>
      <TR VALIGN="baseline">
        <TD WIDTH="100">#LSDateFormat(ExpenseDate)#</TD>
        <TD WIDTH="250">#MovieTitle#</TD>
        <TD WIDTH="150"><EM>#Description#</EM></TD>
        <TD WIDTH="100">#LSCurrencyFormat(ExpenseAmount)#</TD>
      </TR>
    </CFOUTPUT>
  </CFLOOP>

  <!--- Row at bottom of table, after rows of data --->
  <TR>
    <TD WIDTH="100"></TD>
    <TD WIDTH="250"></TD>
    <TD WIDTH="150"></TD>
    <TD WIDTH="100" ALIGN="right">
      <!--- Provide Next/Back links --->
      <CFINCLUDE TEMPLATE="NextNIncludeBackNext.cfm">
    </TD>
  </TR>
</TABLE>


</BODY>
</HTML>
```

Note

This listing relies on the STARTROW and ENDROW attributes for the <CFLOOP> tag. See Chapter 10, "CFML Basics," and Appendix A, "ColdFusion Tag Reference," for detailed information about <CFLOOP>.

First, a query named GetExp is run, which retrieves all expense records from the Expenses table, along with the associated MovieTitle for each expense. The records are returned in reverse date order (most recent expenses first). Next, a variable called RowsPerPage is set to the number of rows that should be displayed to the user at one time. Of course, this value can be adjusted to 20, 50, or whatever you feel is appropriate.

Tip

You could set the RowsPerPage variable in Application.cfm if you wanted to use the same value in a number of different Next N interfaces throughout your application.

The URL.StartRow parameter is established via the <CFPARAM> tag and given a default value of 1 if it is not actually supplied in the URL. Then, a TotalRows variable is set to the number of rows returned by the GetExp query.

Tip

Sometimes it's worth setting a variable just to keep your code clear. In this template, you could skip the <CFSET> for the TotalRows variable and just use GetExp.RecordCount in its place throughout the rest of the code. But the name of the TotalRows variable helps make the role of the value easier to understand, and virtually no performance penalty will exist for the extra line of code.

Next, a variable called EndRow is calculated, which determines which row should be the last one to appear on a given page. In general, the EndRow is simply RowsPerPage past the StartRow. However, the EndRow should never go past the total number of rows in the query, so the Min function is used to ensure that the value is never greater than TotalRows. This becomes important when the user reaches the last page of search results. The URL.StartRow and EndRow values are passed to the STARTROW and ENDROW attributes of the <CFLOOP> that displays the expense records, effectively throttling the display so it shows only the appropriate records for the current page.

StartRowNext and StartRowBack represent what the new StartRow value should be if the user clicks the Next or Back link. If the user clicks Next, the page is reloaded at one row past the current EndRow. If the user clicks Back, the display moves back by the value stored in RowsPerPage (which is 10 in this example).

After this small set of variables has been calculated, the rest of the template is really quite simple. An HTML table is used to display the expense results. The first row of the table displays a message about which rows are currently being shown. It also displays Next and Back links, as appropriate, by including the NextNIncludeBackNext.cfm template (see Listing 23.3). The next row of the table displays some simple column headings. Then, the

<CFLOOP> tag is used to output a table row for each record returned by the GetExp query, but only for the rows from URL.StartRow through EndRow. Finally, the last row of the HTML table repeats the Next and Back links under the expense records, using a second <CFINCLUDE> tag.

Note

For now, don't worry about the fact that the query must be rerun each time the user clicks the Next or Back link. ColdFusion's query-caching feature can be used to ensure that your database is not queried unnecessarily. See Chapter 24, "Improving Performance," for details.

The Application.cfm file shown in Listing 23.2 establishes the REQUEST.DataSource and REQUEST.CompanyName variables used in Listing 23.1. Because they are set in the special REQUEST scope, these variables are available for use within any of this folder's templates, including any custom tags.

This is an excellent way to establish global settings for an application, such as data source names, and is used in most of the Application.cfm templates in the second half of this book. For more information about this use of the special REQUEST scope, see the section "The REQUEST Scope" in Chapter 22, "Building Reusable Components."

LISTING 23.2 Application.cfm—PROVIDING APPLICATION SETTINGS FOR THIS CHAPTER'S EXAMPLES

```
<!---
  Filename:     Application.cfm
  Created by:   Nate Weiss (NMW)
  Date Created: 2/18/2001
  Please Note:  Executes for each page request
--->

<!--- Any variables set here can be used by all our pages --->
<CFSET REQUEST.DataSource = "ows">
<CFSET REQUEST.CompanyName = "Orange Whip Studios">
```

ADDING NEXT AND BACK BUTTONS

Listing 23.3 provides the code that includes the Back and Next links above and below the expense records. The idea here is simple: to show Back and Next links when appropriate. The Back link should be shown whenever the StartRowBack value is greater than 0, which should always be the case unless the user is looking at the first page of records. The Next link should be shown as long as the StartRowNext value is not after the last row of the query, which would be the case only when the user is at the last page of records.

LISTING 23.3 NextNIncludeBackNext.cfm—INCLUDING BACK AND NEXT BUTTONS

```
<!--- Provide Next/Back links --->
<CFOUTPUT>
  <!--- Show link for Back, if appropriate --->
```

LISTING 23.3 CONTINUED

```
  <CFIF StartRowBack GT 0>
    <A HREF="#CGI.SCRIPT_NAME#?StartRow=#StartRowBack#">
      <IMG SRC="../images/BrowseBack.gif" WIDTH="40" HEIGHT="16"
        ALT="Back #RowsPerPage# Records" BORDER="0"></A>
  </CFIF>
  <!--- Show link for Next, if appropriate --->
  <CFIF StartRowNext LT TotalRows>
    <A HREF="#CGI.SCRIPT_NAME#?StartRow=#StartRowNext#">
      <IMG SRC="../images/BrowseNext.gif" WIDTH="40" HEIGHT="16"
        ALT="Next #RowsPerPage# Records" BORDER="0"></A>
  </CFIF>
</CFOUTPUT>
```

As you can see, the Next and Back links always reload the current page, passing the appropriate `StartRow` parameter in the URL. Now the user can navigate through the all the query's records in digestible groups of 10. Figure 23.1 shows what the results look like in a browser.

Note

Because the `CGI.SCRIPT_NAME` variable is used for the Back and Next links, this code continues to provide the correct links even if you change the filename for Listing 23.1. If you find this confusing, you could replace the `CGI.SCRIPT_NAME` with the name of the template the user will be accessing (in this case, `NextN1.cfm`). See Appendix C, "Special ColdFusion Variables and Result Codes," for more information about this handy CGI variable.

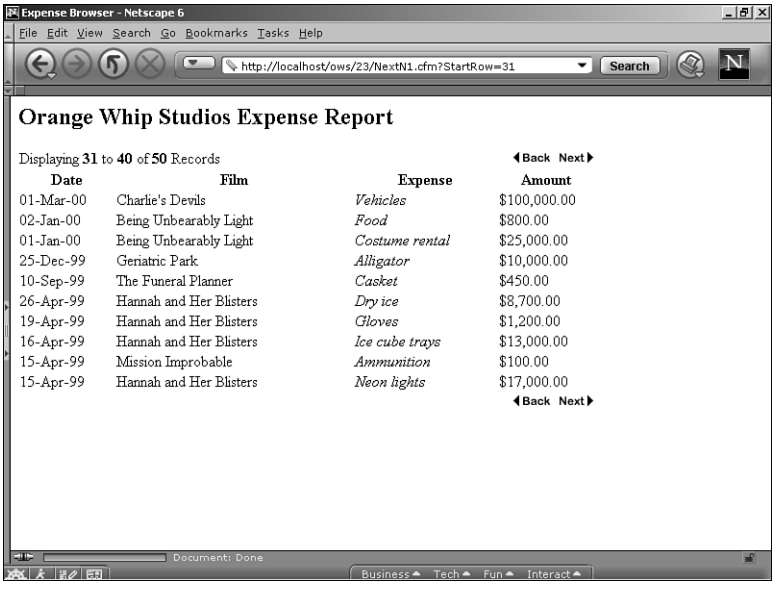

Figure 23.1
Creating a simple Next 10 type of interface for your users is easy.

ALTERNATING ROW COLORS FOR READABILITY

Listing 23.4 is a revised version of Listing 23.1. This version just adds some basic formatting via CSS syntax and presents the rows of data with alternating colors, as shown in Figure 23.2.

LISTING 23.4 `NextN2.cfm`—ADDING CSS-BASED FORMATTING TO THE NEXT N INTERFACE

```
<!--- Retrieve expense records from database --->
<CFQUERY NAME="GetExp" DATASOURCE="#REQUEST.DataSource#">
  SELECT
    f.FilmID, f.MovieTitle,
    e.Description, e.ExpenseAmount, e.ExpenseDate
  FROM
    Expenses e INNER JOIN Films f
    ON e.FilmID = f.FilmID
  ORDER BY
    e.ExpenseDate DESC
</CFQUERY>


<!--- Number of rows to display per Next/Back page  --->
<CFSET RowsPerPage = 10>
<!--- What row to start at? Assume first by default --->
<CFPARAM NAME="URL.StartRow" DEFAULT="1" TYPE="numeric">

<!--- We know the total number of rows from query   --->
<CFSET TotalRows    = GetExp.RecordCount>
<!--- Last row is 10 rows past the starting row, or --->
<!--- total number of query rows, whichever is less --->
<CFSET EndRow       = Min(URL.StartRow + RowsPerPage - 1, TotalRows)>
<!--- Next button goes to 1 past current end row  --->
<CFSET StartRowNext = EndRow + 1>
<!--- Back button goes back N rows from start row --->
<CFSET StartRowBack = URL.StartRow - RowsPerPage>


<!--- Page Title --->
<HTML>
<HEAD><TITLE>Expense Browser</TITLE></HEAD>
<BODY>
<CFOUTPUT><H2>#REQUEST.CompanyName# Expense Report</H2></CFOUTPUT>

<!--- Simple style sheet for formatting --->
<STYLE>
  TH       {font-family:sans-serif;font-size:smaller;
            background:navy;color:white}
  TD       {font-family:sans-serif;font-size:smaller}
  TD.DataA {background:silver;color:black}
  TD.DataB {background:lightgrey;color:black}
</STYLE>

<TABLE WIDTH="600" BORDER="0" CELLSPACING="0" CELLPADDING="1">
  <!--- Row at top of table, above column headers --->
  <TR>
    <TD WIDTH="500" COLSPAN="3">
```

LISTING 23.4 CONTINUED

```
      <!--- Message about which rows are being displayed --->
      <CFOUTPUT>
        Displaying <B>#URL.StartRow#</B> to <B>#EndRow#</B>
        of <B>#TotalRows#</B> Records<BR>
      </CFOUTPUT>
    </TD>
    <TD ALIGN="right">
      <!--- Provide Next/Back links --->
      <CFINCLUDE TEMPLATE="NextNIncludeBackNext.cfm">
    </TD>
  </TR>

  <!--- Row for column headers --->
  <TR>
    <TH WIDTH="100">Date</TH>
    <TH WIDTH="250">Film</TH>
    <TH WIDTH="150">Expense</TH>
    <TH WIDTH="100">Amount</TH>
  </TR>

  <!--- For each query row that should be shown now --->
  <CFLOOP QUERY="GetExp" StartRow="#URL.StartRow#" ENDROW="#EndRow#">
    <!--- Use class "DataA" or "DataB" for alternate rows --->
    <CFSET Class = IIF(GetExp.CurrentRow MOD 2 EQ 0, "'DataA'", "'DataB'")>

    <CFOUTPUT>
      <TR VALIGN="baseline">
        <TD CLASS="#Class#" WIDTH="100">#LSDateFormat(ExpenseDate)#</TD>
        <TD CLASS="#Class#" WIDTH="250">#MovieTitle#</TD>
        <TD CLASS="#Class#" WIDTH="150"><I>#Description#</I></TD>
        <TD CLASS="#Class#" WIDTH="100">#LSCurrencyFormat(ExpenseAmount)#</TD>
      </TR>
    </CFOUTPUT>
  </CFLOOP>

  <!--- Row at bottom of table, after rows of data --->
  <TR>
    <TD WIDTH="100"></TD>
    <TD WIDTH="250"></TD>
    <TD WIDTH="150"></TD>
    <TD WIDTH="100" ALIGN="right">
      <!--- Provide Next/Back links --->
      <CFINCLUDE TEMPLATE="NextNIncludeBackNext.cfm">
    </TD>
  </TR>
</TABLE>


</BODY>
</HTML>
```

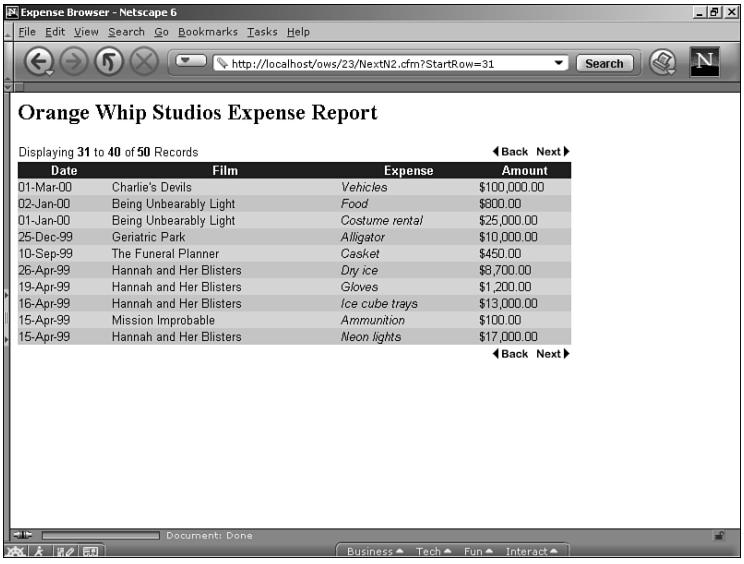

Figure 23.2
The background colors of table cells can be alternated to make the display easier to read.

DEFINING STYLES

The `<STYLE>` block in Listing 23.4 specifies that all `<TH>` cells should be displayed with white lettering on a navy background. Also, two style classes for `<TD>` cells are defined, called `DataA` and `DataB`. By displaying alternate rows with these two classes, the expenses are displayed with alternating background colors, as shown in Figure 23.2.

Inside the `<CFLOOP>` tag, the code alternates between the `DataA` and `DataB` style classes by using ColdFusion's `MOD` operator. The `MOD` operator simply returns the *modulus* of two numbers, which is the remainder left over when the first number is divided by the second. When the `CurrentRow` is an even number, dividing it by `2` results in a remainder of `0`, so the `Class` variable is set to `DataA`. Otherwise, `Class` is set to `DataB`. The `Class` variable is then used as the `CLASS` attribute for the `<TD>` cells that display each row of expenses. The result is the pleasant-looking rendition of the Next N interface shown in Figure 23.2.

Tip

The `DataA` and `DataB` style classes could vary in more than just background color. They could use different typefaces, font colors, bolding, and so on. See a CSS reference for details.

Note

If you don't want to use CSS-based formatting, you could use the `IIF()` test in Listing 23.4 to switch between two color names instead of class names. Then, you would feed the result to the `BGCOLOR` attribute of the `<TD>` tags, instead of the `CLASS` attribute. This would ensure that the rows displayed with alternating colors, even for browsers that don't support CSS (CSS support appeared in Version 4.0 of Internet Explorer and Netscape Communicator). Of course, you could also choose to alternate both the `CLASS` and `BGCOLOR` values.

A NOTE ON THE USE OF IIF()

The line that sets the Class attribute uses the IIF() function, which enables you to choose between two expressions depending on a condition. The IIF() function is comparable to the ? and : operators used in JavaScript and some other languages. The first parameter is the condition, the second parameter determines what the result should be when the condition is True, and the third is what the result should be when condition is False.

When IIF() is used to switch between two strings, as shown previously in Listing 23.4, the second and third parameters must have two sets of quotes because they each will be evaluated as expressions. If the second parameter were written as "DataA" instead of "'DataA'", an error would result because ColdFusion would try to return the value of a variable called DataA, which does not exist. The inner set of single quotation marks tells ColdFusion that the literal string DataA should be returned. For details, see IIF() in Appendix B, "ColdFusion Function Reference."

The IIF() function is used here because it often improves code readability in cases such as this, due to its brevity. If, however, you find it confusing, you could achieve the same result by replacing the single <CFSET> line with this:

```
<CFIF GetExp.CurrentRow MOD 2 EQ 0>
  <CFSET Class = "DataA">
<CFELSE>
  <CFSET Class = "DataB">
</CFIF>
```

LETTING THE USER BROWSE PAGE-BY-PAGE

Many Next N interfaces you see on the Web provide numbered page-by-page links in addition to the customary Back and Next links. If there are 50 records to display, and 10 records are shown per page, the user can use links labeled 1–5 to jump to a particular set of 10 records. This not only gives the user a way to move through the records quickly, but the collection of clickable page numbers also serves as a visual cue or meter that provides a sense of how many records there are to look through.

For clarity, the page-by-page links are implemented in a separate file called NextNIncludePageLinks.cfm. Listing 23.5 shows the code for this new file.

LISTING 23.5 NextNIncludePageLinks.cfm—CREATING PAGE-BY-PAGE LINKS FOR BROWSING RECORDS

```
<!--- Simple "Page" counter, starting at first "Page" --->
<CFSET ThisPage = 1>

<!--- Loop thru row numbers, in increments of RowsPerPage --->
<CFLOOP FROM="1" TO="#TotalRows#" STEP="#RowsPerPage#" INDEX="PageRow">
  <!--- Detect whether this "Page" currently being viewed --->
  <CFSET IsCurrentPage = (PageRow GTE URL.StartRow) AND (PageRow LTE EndRow)>

  <!--- If this "Page" is current page, show without link --->
  <CFIF IsCurrentPage>
    <CFOUTPUT><B>#ThisPage#</B></CFOUTPUT>
```

```
  <!--- Otherwise, show with link so user can go to page  --->
  <CFELSE>
    <CFOUTPUT>
      <A HREF="#CGI.SCRIPT_NAME#?StartRow=#PageRow#">#ThisPage#</A>
    </CFOUTPUT>
  </CFIF>

  <!--- Increment ThisPage variable --->
  <CFSET ThisPage = ThisPage + 1>
</CFLOOP>
```


Similar to the Back and Next code shown in Listing 23.3, this template's job is to generate a number of links that reload the current template, passing the appropriate `StartRow` parameter in the URL.

First, a variable named `ThisPage` is set to `1`. This variable is incremented as each page-by-page link is displayed. Next, a `<CFLOOP>` tag is used to create each page-by-page link. Because the `STEP` attribute is set to the value of `RowsPerPage`, the `PageRow` variable increments by `10` for each iteration of the loop, until it exceeds `TotalRows`. So, the first time through the loop, `ThisPage` and `PageRow` are both `1`. The second time through the loop, `ThisPage` is `2` and `PageRow` is `11`, and so on.

The next `<CFSET>` determines whether the user is already looking at the page of results currently being considered by the loop. If the current value of `PageRow` is between the `StartRow` and `EndRow` values (refer to Listing 23.4), `IsCurrentPage` is `True`. Now the page number can be displayed by outputting the value of `ThisPage`. If `ThisPage` is the page currently being viewed, it is shown in boldface. If not, the page number is presented as a link to the appropriate page by passing the value of `PageRow` in the URL as the `StartRow` parameter. Now the user can see where she is in the records by looking for the boldface number, and she can jump to other pages by clicking the other numbers, as shown in Figure 23.3.

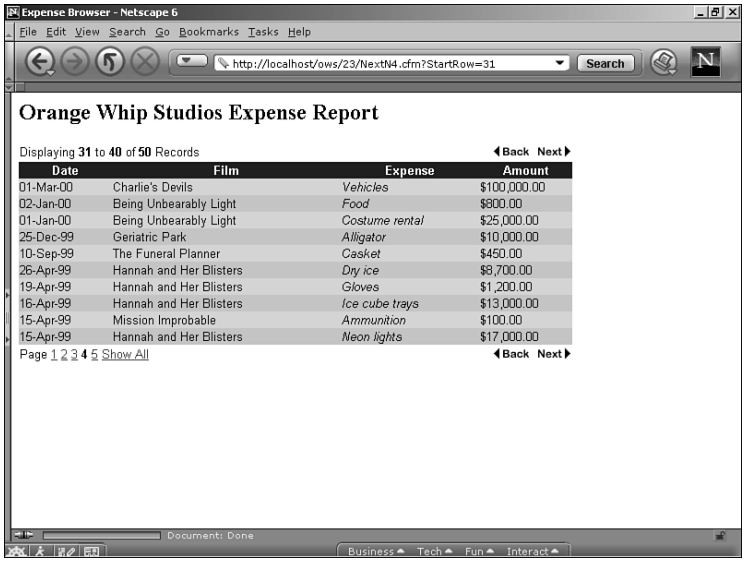

Figure 23.3
The completed interface includes Back, Next, Page-by-Page, and Show All links for easy navigation.

Now that the code has been written, it can be included in the Next N interface with a simple <CFINCLUDE> tag, like so:

```
<!--- Shortcut links for "Pages" of search results --->
Page <CFINCLUDE TEMPLATE="NextNIncludePageLinks.cfm">
```

Listing 23.6 (on this book's CD-ROM) builds on the previous version (refer to Listing 23.4) by adding the <CFINCLUDE> tag shown previously. The code is otherwise unchanged.

You can also see this <CFINCLUDE> tag in the next version of this template (see Listing 23.7).

ADDING A SHOW ALL OPTION

Although Next N interfaces are great for keeping your pages from getting too large to navigate properly, your users sometimes might need a way to see all the records at once (for instance, when they need to print a hard copy). Therefore, it's worth considering the addition of a Show All link, which essentially serves to override the Next N interface if the user so desires.

Listing 23.7 is the final version of the Next N interface, which now includes a Show All option, as well as the page-by-page navigation included in the previous version. As you can see, only a few lines of new code were necessary to put the Show All option into place.

LISTING 23.7 NextN4.cfm—ADDING A WAY TO VIEW ALL RECORDS AT ONCE

```
<!--- Retrieve expense records from database --->
<CFQUERY NAME="GetExp" DATASOURCE="#REQUEST.DataSource#">
  SELECT
    f.FilmID, f.MovieTitle,
    e.Description, e.ExpenseAmount, e.ExpenseDate
  FROM
    Expenses e INNER JOIN Films f
    ON e.FilmID = f.FilmID
  ORDER BY
    e.ExpenseDate DESC
</CFQUERY>


<!--- Number of rows to display per Next/Back page  --->
<CFSET RowsPerPage = 10>
<!--- What row to start at? Assume first by default --->
<CFPARAM NAME="URL.StartRow" DEFAULT="1" TYPE="numeric">
<!--- Allow for Show All parameter in the URL --->
<CFPARAM NAME="URL.ShowAll" TYPE="boolean" DEFAULT="No">

<!--- We know the total number of rows from query   --->
<CFSET TotalRows    = GetExp.RecordCount>
<!--- Show all on page if ShowAll is passed in URL   --->
<CFIF URL.ShowAll>
  <CFSET RowsPerPage = TotalRows>
</CFIF>
<!--- Last row is 10 rows past the starting row, or --->
<!--- total number of query rows, whichever is less --->
<CFSET EndRow       = Min(URL.StartRow + RowsPerPage - 1, TotalRows)>
<!--- Next button goes to 1 past current end row  --->
```

```
<CFSET StartRowNext = EndRow + 1>
<!--- Back button goes back N rows from start row --->
<CFSET StartRowBack = URL.StartRow - RowsPerPage>


<!--- Page Title --->
<HTML>
<HEAD><TITLE>Expense Browser</TITLE></HEAD>
<BODY>
<CFOUTPUT><H2>#REQUEST.CompanyName# Expense Report</H2></CFOUTPUT>

<!--- Simple style sheet for formatting --->
<STYLE>
  TH       {font-family:sans-serif;font-size:smaller;
            background:navy;color:white}
  TD       {font-family:sans-serif;font-size:smaller}
  TD.DataA {background:silver;color:black}
  TD.DataB {background:lightgrey;color:black}
</STYLE>

<TABLE WIDTH="600" BORDER="0" CELLSPACING="0" CELLPADDING="1">
  <!--- Row at top of table, above column headers --->
  <TR>
    <TD WIDTH="500" COLSPAN="3">
      <!--- Message about which rows are being displayed --->
      <CFOUTPUT>
        Displaying <B>#URL.StartRow#</B> to <B>#EndRow#</B>
        of <B>#TotalRows#</B> Records<BR>
      </CFOUTPUT>
    </TD>
    <TD WIDTH="100" ALIGN="right">
      <CFIF NOT URL.ShowAll>
        <!--- Provide Next/Back links --->
        <CFINCLUDE TEMPLATE="NextNIncludeBackNext.cfm">
      </CFIF>
    </TD>
  </TR>

  <!--- Row for column headers --->
  <TR>
    <TH WIDTH="100">Date</TH>
    <TH WIDTH="250">Film</TH>
    <TH WIDTH="150">Expense</TH>
    <TH WIDTH="100">Amount</TH>
  </TR>

  <!--- For each query row that should be shown now --->
  <CFLOOP QUERY="GetExp" StartRow="#URL.StartRow#" ENDROW="#EndRow#">
    <!--- Use class "DataA" or "DataB" for alternate rows --->
    <CFSET Class = IIF(GetExp.CurrentRow MOD 2 EQ 0, "'DataA'", "'DataB'")>

    <CFOUTPUT>
      <TR VALIGN="baseline">
        <TD CLASS="#Class#" WIDTH="100">#LSDateFormat(ExpenseDate)#</TD>
        <TD CLASS="#Class#" WIDTH="250">#MovieTitle#</TD>
        <TD CLASS="#Class#" WIDTH="150"><I>#Description#</I></TD>
        <TD CLASS="#Class#" WIDTH="100">#LSCurrencyFormat(ExpenseAmount)#</TD>
      </TR>
```

LISTING 23.7 CONTINUED

```
    </CFOUTPUT>
  </CFLOOP>

  <!--- Row at bottom of table, after rows of data --->
  <TR>
    <TD WIDTH="500" COLSPAN="3">
      <CFIF NOT URL.ShowAll>
        <!--- Shortcut links for "Pages of search results --->
        Page <CFINCLUDE TEMPLATE="NextNIncludePageLinks.cfm">
        <!--- Show All link --->
        <CFOUTPUT>
          <A HREF="#CGI.SCRIPT_NAME#?ShowAll=Yes">Show All</A>
        </CFOUTPUT>
      </CFIF>
    </TD>
    <TD WIDTH="100" ALIGN="right">
      <CFIF NOT URL.ShowAll>
        <!--- Provide Next/Back links --->
        <CFINCLUDE TEMPLATE="NextNIncludeBackNext.cfm">
      </CFIF>
    </TD>
  </TR>
</TABLE>


</BODY>
</HTML>
```

Near the top of the template, a new URL parameter called `ShowAll` is introduced and given a default value of `No`. Two lines later, a simple `<CFIF>` test is used to set the `RowsPerPage` variable to the value of `TotalRows` if `URL.ShowAll` is `True`. Therefore, if the page is accessed with `ShowAll=Yes` in the URL, the page displays all the records in one large group.

The rest of the template is mostly unchanged from the last version. The only difference is the addition of a few `<CFIF>` tests throughout, so the Back, Next, and page-by-page links are not shown when in show all mode. The final results are shown in Figure 23.3.

Note

Again, for now, don't worry about the fact that the query must be rerun each time the user clicks Next, Back, Show All, or one of the numbered page links. ColdFusion's query-caching feature can be used to ensure that your database is not queried unnecessarily. See Chapter 24, "Improving Performance," for details.

RETURNING PAGE OUTPUT RIGHT AWAY WITH CFFLUSH

By default, the output of all ColdFusion templates is automatically *buffered* by the server, which means the HTML for the entire page is sent to the browser at once, after all processing has been completed. In general, this isn't a problem. In fact, it enables

ColdFusion to pull off a number of cool tricks internally (see the section "When You Can't Flush the Buffer," later in this chapter).

That said, in some instances you will want ColdFusion to return the HTML it generates right away, as the template is executing. The browser will be capable of receiving and displaying the HTML as it is generated. Meanwhile, ColdFusion can be finishing the remainder of the template. The act of telling ColdFusion to send back the generated output right away is called *clearing the page buffer*.

WHEN TO CLEAR THE BUFFER

The two basic situations in which you might want to clear the page buffer are as follows:

- **Large Pages**—If the template you are working on will output a lot of information, such as a long article all on one page, or some type of report that will have many records, you might want to flush the page buffer after every 1,000 characters of HTML have been generated. This causes the page to appear to display more quickly because the user can start reading the page before it has been completely received. It can also be easier on the ColdFusion server because the entire page will never have to be in its RAM at the same time.
- **Long-Running Pages**—Sometimes one of your templates might need to perform some type of operation that is inherently slow, but that doesn't necessarily output a large amount of HTML. For instance, if the user is placing an order, verifying his credit card number might take 10 or 15 seconds (see Chapter 29, "Online Commerce"). By clearing the page buffer several times during the order process, you can display a series of "please wait" messages so the user can see that something is actually happening.

In both of the previous situations, clearing the page buffer judiciously can make your applications appear to be more responsive because they give more feedback to the user sooner. That can mean a better user experience.

Caution

You should not just start clearing the page buffer regularly in all your ColdFusion templates. In particular, it is *not* recommended that you place a `<CFFLUSH>` tag in your `Application.cfm` file. See the section "When You Can't Flush the Buffer," later in this chapter.

THE EXCEPTION, NOT THE RULE

Most ColdFusion pages don't fall into either of the categories discussed previously. That is, most of your application's templates will not generate tons and tons of HTML code, and most of them will complete their executions normally in well under a second.

So, clearing the page buffer usually doesn't have much impact on the average ColdFusion template. And after the page buffer has been cleared, a number of features can no longer be used and will generate error messages (see the section "When You Can't Flush the Buffer," later in this chapter).

In short, the capability to flush the page buffer is helpful for dealing with certain special situations, as outlined previously. Unless the template you are working on will produce a large amount of output or will take a long time to process, just let ColdFusion buffer the page normally.

INTRODUCING THE <CFFLUSH> TAG

ColdFusion 5.0 provides a new tag called <CFFLUSH> that lets you clear the server's page buffer programmatically. As soon as ColdFusion encounters a <CFFLUSH> tag, it sends anything the template has generated so far to the browser. If the browser can, it displays that content to the user while ColdFusion continues working on the template.

The <CFFLUSH> tag takes just one attribute—INTERVAL—which is optional. You can use <CFFLUSH> without INTERVAL; that simply causes the page buffer to be flushed at the moment the tag is encountered. If you provide a number to INTERVAL, ColdFusion continues to flush the page cache whenever that many bytes have been generated by your template. So, INTERVAL="1000" causes the page buffer to be cleared after every 1,000 bytes, which would mean after every 1,000th character or so.

Tip

If you're not familiar with what a *byte* is, don't worry about it. Just think of the INTERVAL attribute as specifying a number of characters, rather than a number of bytes. Basically, each character in the normal English character set takes up a byte in a computer's memory.

FLUSHING THE OUTPUT BUFFER FOR LARGE PAGES

Consider the Show All option shown in Listing 23.7, earlier in this chapter. Over time, hundreds or thousands of records could exist in the Expenses table, causing the Show All display to become extremely large. Therefore, it becomes a good candidate for <CFFLUSH> tag.

For instance, near the top of Listing 23.7, you could change this code:

```
<!--- Show all on page if ShowAll is passed in URL   --->
<CFIF URL.ShowAll>
  <CFSET RowsPerPage = TotalRows>
</CFIF>
```

to this:

```
<!--- Show all on page if ShowAll is passed in URL   --->
<CFIF URL.ShowAll>
  <CFSET RowsPerPage = TotalRows>

  <!--- Flush the page buffer every 5,000 characters --->
  <CFFLUSH INTERVAL="5000">
</CFIF>
```

Now, the page should begin to be sent to the user's browser in 5,000-character chunks, instead of all at once. The result is that the page should display more quickly when the Show All option is used. Note, however, that the difference might not be particularly

noticeable until the Expenses table starts to get quite large. Even then, the difference will likely be more noticeable for modem users.

FLUSHING THE OUTPUT BUFFER FOR LONG-RUNNING PROCESSES

You already have seen how <CFFLUSH> can help with templates that return large pages to the browser. You also can use <CFFLUSH> to help deal with situations in which a lengthy process needs to take place (such as verifying and charging a user's credit card or executing a particularly complex record-updating process).

SIMULATING A LONG-RUNNING PROCESS

To keep the examples in this chapter simple, the following code snippet is used to simulate some type of time-consuming process. Because this code should never be used in an actual, real-world application, it is not explained in detail here. The basic idea is to create a <CFLOOP> that keeps looping over and over again until a specified number of seconds have passed.

For instance, this will force ColdFusion to spin its wheels for five seconds:

```
<CFSET InitialTime = Now()>
<CFLOOP CONDITION="DateDiff('s', InitialTime, Now()) LT 5"></CFLOOP>
```

See Appendix B for more information about the Now() and DateDiff() functions.

Caution

The previous code snippet is a very inefficient way to cause ColdFusion to pause for a specified amount of time and should not be used in your own production code. It will cause ColdFusion to hog the CPU during the time period specified. It is used in this chapter only as a placeholder for whatever time-consuming process you might need to execute in your own templates.

Tip

The <CF_WaitFor> custom tag, available from http://www.nateweiss.com, can be used to safely cause ColdFusion to wait for a specified number of seconds on Windows systems. It will not, however, work on ColdFusion servers running on other platforms.

DISPLAYING A PLEASE-WAIT TYPE OF MESSAGE

The FlushTest.cfm template shown in Listing 23.8 demonstrates how you can use the <CFFLUSH> tag to output page content before and after a lengthy process. Here, the user is asked to wait while an order is processed.

LISTING 23.8 FlushTest.cfm—DISPLAYING MESSAGES BEFORE AND AFTER A LENGTHY PROCESS

```
<HTML>
<HEAD><TITLE>&lt;CFFLUSH&gt; Example</TITLE></HEAD>
<BODY>
```

LISTING 23.8 CONTINUED

```
  <!--- Initial message --->
  <P><STRONG>Please Wait</STRONG><BR>
  We are processing your order.<BR>
  This process may take up to several minutes.<BR>
  Please do not reload or leave this page until the process is complete.<BR>

  <!--- Flush the page output buffer --->
  <!--- The above code is sent to the browser right now --->
  <CFFLUSH>

  <!--- Time-consuming process goes here --->
  <!--- Here, ColdFusion is forced to wait for 5 seconds --->
  <!--- Do not use this CFLOOP technique in actual code! --->
  <CFSET InitialTime = Now()>
  <CFLOOP CONDITION="DateDiff('s', InitialTime, Now()) LT 5"></CFLOOP>

  <!--- Display "Success" message --->
  <P><STRONG>Thank You.</STRONG><BR>
  Your order has been processed.<BR>

</BODY>
</HTML>
```

As you can see, the code is very simple. First, a "please wait" message is displayed, using ordinary HTML tags. Then, the `<CFFLUSH>` tag is used to flush the page buffer, enabling the user to see the message immediately. Next, the time-consuming process is performed (you would replace the `<CFLOOP>` snippet with whatever is appropriate for your situation). The rest of the page can then be completed normally.

If you visit this template with your browser, you should see the please wait message alone on the page at first. After about five seconds, the thank you message will appear. This gives your application a more responsive feel for your users.

DISPLAYING A GRAPHICAL PROGRESS METER

With the help of some simple JavaScript code, you can create a graphical progress meter while a particularly long process executes. The code shown in Listing 23.9 is similar to the previous listing, except that it assumes the time-consuming process the template needs to accomplish has several steps. When the page first appears, it shows an image of a progress indicator that reads `0%`. As each step of the lengthy process is completed, the image is updated so the indicator reads `25%`, `50%`, `75%`, and finally `100%`.

LISTING 23.9 `FlushMeter.cfm`—DISPLAYING A PROGRESS METER BY SWAPPING IMAGES VIA JAVASCRIPT

```
<HTML>
<HEAD><TITLE>&lt;CFFLUSH&gt; Example</TITLE></HEAD>
<BODY>
```

```
  <!--- Initial Message --->
  <P><STRONG>Please Wait</STRONG><BR>
  We are processing your order.<BR>

  <!--- Create the "Meter" image object --->
  <!--- Initially, it displays a blank GIF --->
  <IMG NAME="Meter" SRC="../images/PercentBlank.gif"
    WIDTH="200" HEIGHT="16" ALT="" BORDER="0">

  <!--- Flush the page buffer --->
  <CFFLUSH>

  <!--- Loop from 0 to 25 to 50 to 75 to 100 --->
  <CFLOOP FROM="0" TO="100" STEP="25" INDEX="i">
    <!--- Time-consuming process goes here --->
    <!--- Here, ColdFusion waits for 5 seconds as an example --->
    <!--- Do not use this technique in actual code! --->
    <CFSET InitialTime = Now()>
    <CFLOOP CONDITION="DateDiff('s', InitialTime, Now()) LT 2"></CFLOOP>

    <!--- Change the SRC attribute of the Meter image --->
    <CFOUTPUT>
      <SCRIPT LANGUAGE="JavaScript">
        document.images["Meter"].src = '../images/Percent#i#.gif';
      </SCRIPT>
    </CFOUTPUT>
    <CFFLUSH>
  </CFLOOP>

  <!--- Display "Success" message --->
  <P><STRONG>Thank You.</STRONG><BR>
  Your order has been processed.<BR>
</BODY>
</HTML>
```

First, an ordinary <IMG> tag is used to put the progress indicator on the page. The image's SRC attribute is set to the PercentBlank.gif image, which is just an empty, transparent (*spacer*) image that won't show up (except as empty space) on the page. The <CFFLUSH> tag is used to ensure that the browser receives the <IMG> tag code and displays the placeholder image right away.

Next, the <CFLOOP> tag is used to simulate some type of time-consuming, five-step process. Because of the STEP attribute, the value of i is 0 the first time through the loop, then 25, then 50, then 75, and then 100. Each time through the loop, a <SCRIPT> tag is output that contains JavaScript code to change the src property of the meter <IMG>, which causes the meter effect. The buffer is flushed with <CFFLUSH> after each <SCRIPT> tag, so the browser can receive and execute the script right away. The first time through the loop, the <IMG> is set to display the Percent0.gif file, then Percent25.gif, and so on. The end result is a simple progress meter that can help your users feel like they are still connected during whatever time-consuming processes they initiate. Figure 23.4 shows what the meter looks like in a browser.

Note

Full coverage of the use of an image's `src` property to swap the images it displays can't be included here. For more information, consult a JavaScript reference book, or the scripting reference section under HTML Reference in ColdFusion Studio's online help.

If the user's browser doesn't support JavaScript, the `<IMG>` tag will simply continue to display the `PercentBlank.gif` image, which the user won't even notice because it is invisible.

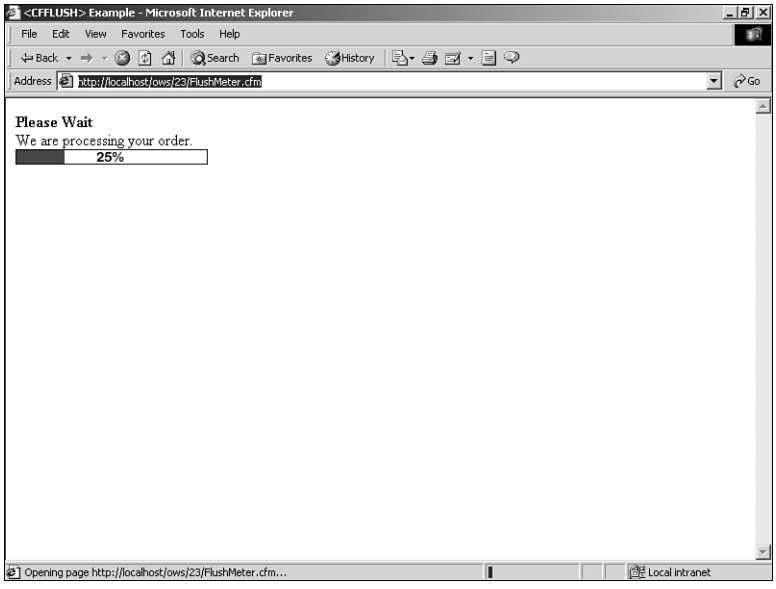

Figure 23.4
The `<CFFLUSH>` tag enables you to display progress indicators during lengthy processes.

FLUSHING THE OUTPUT BUFFER BETWEEN TABLE ROWS

In Listing 23.7, a Show All link was added to the Next N interface for browsing Orange Whip Studio's expenses. Depending on the number of rows in the `Expenses` table, that page could produce quite a bit of output. You might want to consider flushing the page output buffer after every few rows of data, so the user will start to see the rows while the page is being generated.

Listing 23.10 shows how you can add a `<CFFLUSH>` in the middle of an output loop, so groups of rows get sent back to the browser right away. In this example, the rows are sent back in groups of five; in practice, you might want to choose a higher number, such as 20 or 30 rows.

LISTING 23.10 `NextN5.cfm`—SENDING CONTENT TO THE BROWSER AFTER EVERY FIFTH ROW OF DATA

```
<!--- Retrieve expense records from database --->
<CFQUERY NAME="GetExp" DATASOURCE="#REQUEST.DataSource#">
  SELECT
    f.FilmID, f.MovieTitle,
```

```
    e.Description, e.ExpenseAmount, e.ExpenseDate
  FROM
    Expenses e INNER JOIN Films f
    ON e.FilmID = f.FilmID
  ORDER BY
    e.ExpenseDate DESC
</CFQUERY>


<!--- Number of rows to display per Next/Back page  --->
<CFSET RowsPerPage = 10>
<!--- What row to start at? Assume first by default --->
<CFPARAM NAME="URL.StartRow" DEFAULT="1" TYPE="numeric">
<!--- Allow for Show All parameter in the URL --->
<CFPARAM NAME="URL.ShowAll" TYPE="boolean" DEFAULT="No">

<!--- We know the total number of rows from query   --->
<CFSET TotalRows = GetExp.RecordCount>
<!--- Show all on page if ShowAll passed in URL   --->
<CFIF URL.ShowAll>
  <CFSET RowsPerPage = TotalRows>
</CFIF>
<!--- Last row is 10 rows past the starting row, or --->
<!--- total number of query rows, whichever is less --->
<CFSET EndRow = Min(URL.StartRow + RowsPerPage - 1, TotalRows)>
<!--- Next button goes to 1 past current end row  --->
<CFSET StartRowNext = EndRow + 1>
<!--- Back button goes back N rows from start row --->
<CFSET StartRowBack = URL.StartRow - RowsPerPage>


<!--- Page Title --->
<HTML>
<HEAD><TITLE>Expense Browser</TITLE></HEAD>
<BODY>
<CFOUTPUT><H2>#REQUEST.CompanyName# Expense Report</H2></CFOUTPUT>

<!--- simple style sheet for formatting --->
<STYLE>
  TH       {font-family:sans-serif;font-size:smaller;
            background:navy;color:white}
  TD       {font-family:sans-serif;font-size:smaller}
  TD.DataA {background:silver;color:black}
  TD.DataB {background:lightgrey;color:black}
</STYLE>

<TABLE WIDTH="600" BORDER="0" CELLSPACING="0" CELLPADDING="1">
  <!--- Row at top of table, above column headers --->
  <TR>
    <TD WIDTH="500" COLSPAN="3">
      <!--- Message about which rows are being displayed --->
      <CFOUTPUT>
        Displaying <B>#URL.StartRow#</B> to <B>#EndRow#</B>
        of <B>#TotalRows#</B> Records<BR>
      </CFOUTPUT>
    </TD>
    <TD WIDTH="100" ALIGN="right">
      <CFIF NOT URL.ShowAll>
```

LISTING 23.10 CONTINUED

```
      <!--- Provide Next/Back links --->
      <CFINCLUDE TEMPLATE="NextNIncludeBackNext.cfm">
    </CFIF>
  </TD>
</TR>

<!--- Row for column headers --->
<TR>
  <TH WIDTH="100">Date</TH>
  <TH WIDTH="250">Film</TH>
  <TH WIDTH="150">Expense</TH>
  <TH WIDTH="100">Amount</TH>
</TR>

<!--- For each query row that should be shown now --->
<CFLOOP QUERY="GetExp" StartRow="#URL.StartRow#" ENDROW="#EndRow#">
  <!--- Use class "DataA" or "DataB" for alternate rows --->
  <CFSET Class = IIF(GetExp.CurrentRow MOD 2 EQ 0, "'DataA'", "'DataB'")>

  <!--- Actual data display --->
  <CFOUTPUT>
    <TR VALIGN="baseline">
      <TD CLASS="#Class#" WIDTH="100">#LSDateFormat(ExpenseDate)#</TD>
      <TD CLASS="#Class#" WIDTH="250">#MovieTitle#</TD>
      <TD CLASS="#Class#" WIDTH="150"><I>#Description#</I></TD>
      <TD CLASS="#Class#" WIDTH="100">#LSCurrencyFormat(ExpenseAmount)#</TD>
    </TR>
  </CFOUTPUT>

  <!--- If showing all records, flush the page buffer after every 5th row --->
  <CFIF URL.ShowAll>
    <CFIF GetExp.CurrentRow MOD 5 EQ 0>
      <!--- End the current table --->
      </TABLE>
      <!--- Flush the page buffer --->
      <CFFLUSH>
      <!--- Start a new table --->
      <TABLE WIDTH="600" BORDER="0" CELLSPACING="0" CELLPADDING="1">
      <!--- Simulate a time-intensive process --->
      <CFSET InitialTime = Now()>
      <CFLOOP CONDITION="DateDiff('s', InitialTime, Now()) LT 1"></CFLOOP>
    </CFIF>
  </CFIF>

</CFLOOP>

<!--- Row at bottom of table, after rows of data --->
<TR>
  <TD WIDTH="500" COLSPAN="3">
    <CFIF NOT URL.ShowAll>
      <!--- Shortcut links for "Pages" of search results --->
      Page <CFINCLUDE TEMPLATE="NextNIncludePageLinks.cfm">
      <!--- Show All Link --->
      <CFOUTPUT>
        <A HREF="#CGI.SCRIPT_NAME#?ShowAll=Yes">Show All</A>
      </CFOUTPUT>
```

```
      </CFIF>
    </TD>
    <TD WIDTH="100" ALIGN="right">
      <CFIF NOT URL.ShowAll>
        <!--- Provide Next/Back links --->
        <CFINCLUDE TEMPLATE="NextNIncludeBackNext.cfm">
      </CFIF>
    </TD>
  </TR>
</TABLE>


</BODY>
</HTML>
```


This code listing is mostly unchanged from the previous version (refer to Listing 23.7). The only significant change is the addition of the <CFIF> block at the end of the <CFLOOP> block. The code in this block executes only if the user has clicked the Show All link, and only if the current row number is evenly divisible by 5 (that is, only for every fifth row).

If both of these conditions are true, the current <TABLE> tag (the one opened near the top of the listing) is closed with a closing </TABLE> tag. The page buffer is then flushed using <CFFLUSH>, and a new table is started with an opening <TABLE> tag that matches the one from the top of the listing. In other words, the expense records are shown as a series of five-row tables that are each sent to the browser individually, rather than as one long table that gets sent to the browser at once. Because each of these minitables is complete, with beginning and ending <TABLE> tags, the browser can display them as it receives them (most browsers can't properly render a table until the closing </TABLE> tag has been encountered).

After the page buffer is cleared, this template waits for one second, using the same time-delay technique that was used in the progress meter example (refer to Listing 23.9). Again, you should never use this technique in your actual code templates. It is used here as a simple way of causing ColdFusion to pause for a moment, so you can see the effect of the page flushes.

If you visit Listing 23.10 with your Web browser and click the Show All link, you will see that the rows of data are presented to you in small groups, with a one-second pause between each group. This proves that the buffer is being cleared, and that a user accessing a very long page over a slow connection would at least be able to begin viewing the records before the entire page had been received.

Of course, in practice, you wouldn't have the time-delay loop at all. It is only included here to make the effect easier to see while developing.

WHEN YOU CAN'T FLUSH THE BUFFER

This section has introduced the <CFFLUSH> tag and pointed out several situations in which it can be helpful. However, because it causes the content your templates generate to be sent to the browser in pieces—rather than the whole page at once—certain ColdFusion tags and features that depend on being capable of manipulating the page as a whole cannot be used after a <CFFLUSH> tag.

RESTRICTIONS ON COOKIE USE

After the <CFFLUSH> tag has been used on a page, telling ColdFusion to set a cookie in the browser is no longer possible. This is because cookies are set by sending special HTTP headers to the browser, and all HTTP headers must be sent to the browser before any actual HTML content. So, after a <CFFLUSH> tag has been used, sending any additional headers to the browser is no longer possible, which in turn means that it's too late for ColdFusion to set any cookies.

If you really need to set a cookie after a <CFFLUSH>, you can use JavaScript to set the cookie. For your convenience, a custom tag called <CF_SetCookieViaJS> has been included on the CD-ROM for this book. The custom tag supports three attributes—COOKIENAME, COOKIEVALUE, and EXPIRES—which correspond to the NAME, VALUE, and EXPIRES attributes for the regular <CFCOOKIE> tag. The EXPIRES attribute is optional.

So, instead of

```
<CFCOOKIE
  NAME="MyCookie"
  VALUE="My Value">
```

You would use

```
<CF_SetCookieViaJS
  CookieName="MyCookie"
  CookieValue="My Value">
```

Please note that the cookie will be set only if the user's browser supports JavaScript and if JavaScript has not been disabled.

Caution

This custom tag is not supported and is merely presented as a workaround for situations in which you must set a cookie after a <CFFLUSH> tag. Whenever possible, it is recommended that you set cookies using the usual <CFCOOKIE> and <CFSET> methods explained in Chapter 20.

Tip

If you are somewhat familiar with JavaScript, you could study the SetCookieViaJS.cfm custom tag template (on the CD-ROM) as an example of how custom tags can be used to generate JavaScript code.

Note

The PATH, SECURE, and DOMAIN attributes from <CFCOOKIE> are not supported by this custom tag, but they could easily be added by editing the custom tag template. See Chapter 22 for information about building custom tags.

RESTRICTIONS ON <CFLOCATION>

After a <CFFLUSH> tag has been encountered, you can no longer use the <CFLOCATION> tag to redirect the user to another page. This is because <CFLOCATION> works by sending a redirect header back to the browser. After the first <CFFLUSH> tag has been encountered on a page,

the page's headers have already been sent to the browser; thus, it is too late to redirect the browser to another page using the usual methods provided by HTTP alone.

There are a few workarounds to this problem. Both rely on the browser to interpret your document in a certain way, and they are not part of the standard HTTP protocol. That said, these methods should work fine with most browsers.

The first workaround is to include a <META> tag in the document, with an HTTP-EQUIV attribute set to Refresh. Then, provide the URL for the next page in the CONTENT attribute, as shown in the following. Most browsers interpret this as an instruction to go to the specified page as soon as the tag is encountered.

So, instead of this:

```
<CFLOCATION URL="MyNextPage.cfm">
```

you would use this:

```
<META HTTP-EQUIV="Refresh" CONTENT="0; URL=MyNextPage.cfm">
```

Note

If you want the redirect to occur after five seconds rather than right away, you could change the 0 in the previous snippet to 5. See the topic titled Client Pull in ColdFusion Studio's online help for more information about this use of the <META> tag.

Another workaround is to use JavaScript. The following snippet could also be used in place of the <CFLOCATION> shown previously. However, if JavaScript is disabled or not supported by the client, nothing will happen. See the scripting reference in ColdFusion Studio for more information about this use of the document.location object:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  document.location.href ="MyNextPage.cfm";
//-->
</SCRIPT>
```

OTHER RESTRICTIONS

Several other tags cannot be used after a <CFFLUSH> tag has been encountered, for the same basic reasons the <CFCOOKIE> and <CFLOCATION> tags cannot be used (they all need to write header information).

These tags cannot be used after a <CFFLUSH>:

- <CFCONTENT>
- <CFCOOKIE>
- <CFFORM>
- <CFHEADER>
- <CFHTMLHEAD>
- <CFLOCATION>