CHAPTER 24

Enhancing Forms with Client-Side Java

IN THIS CHAPTER

About ColdFusion's Java-Based Form Controls

In Chapter 11, "ColdFusion Forms"; Chapter 12, "Form Data Validation"; and Chapter 13, "Using Forms to Add or Change Data," you learned a great deal about creating forms for your Web applications. Not only did you learn about using standard HTML forms using <FORM>, <INPUT>, and <SELECT> tags, you also learned how to take advantage of ColdFusion's beefed-up, JavaScript-enriched encapsulations of those tags (<CFFORM>, <CFINPUT>, and <CFSELECT>).

ColdFusion also provides a collection of useful Java-based form controls, which you can use to create forms that present or collect information in ways that aren't possible with standard HTML form controls.

The controls are:

- **Tree Control.** Similar to the tree-based file-navigation pane in the Windows Explorer, the <CFTREE> form control provides a professional-looking, compact, and intuitive way to present hierarchical information to your users.
- **Grid Control.** Somewhat similar to a spreadsheet application, the <CFGRID> form control provides an easy way for your users to view or edit rows and columns of data. You can add check boxes, drop-down lists, and color-coding to the grid.
- **Slider Control.** The <CFSLIDER> control is ideal for situations in which you want to collect a rating, ranking, or some other numeric value that falls within a definite range.
- **Text Input Control.** ColdFusion also provides a Java-based text input applet, which—like the <CFINPUT> tags you learned about in Chapters 12 and 13—can be used to get validated text input from the user. In most cases, <CFINPUT> is better.

About the Java Controls and the Java Plug-In

The Java-based form controls included with ColdFusion are compatible with nearly all versions of the major browsers. You don't need to know anything about Java to use these controls; the deployment of the applets is taken care of by the ColdFusion server.

Before the controls will work on a user's browser, the Java Plug-in must be installed on the user's machine. The Java Plug-in enables Java applets (such as `<CFTREE>`, `<CFGRID>`, and `<CFSLIDER>`) to run using a Java runtime environment provided by the plug-in, rather than the Java virtual machine provided by the browser, if any.

If the Java Plug-in has never been installed, it is installed automatically if the user is using Internet Explorer (IE downloads and installs the plug-in from your ColdFusion server). If the user is using Netscape or some other browser, she might have to download and install the plug-in before the controls will work. For more information about the Java Plug-in, visit `www.javasoft.com/products/plugin`.

NOTE

In either case, you need to be aware that the initial Java Plug-in software is an approximately 5MB download. The download should need to be performed only once per machine, but you might want to consider *not* using these controls in applications geared toward people who might be accessing your pages via a slow connection.

NOTE

ColdFusion 4.5 and earlier supported an `ENABLECAB` attribute for `<CFFORM>`, which caused the `<CFFORM>` Java applets to be available for Internet Explorer users more quickly than for Netscape users because all the Java classes needed by the applets were delivered in a single Windows Cabinet (.cab) file. In ColdFusion MX, the classes are now delivered in a single Java Archive (.jar) file for both IE and Netscape users, which results in good performance for all supported browsers. The `ENABLECAB` attribute is thus no longer necessary. You don't have to remove it from your existing templates, but you should leave it out of any new ones.

Using <CFTREE>

The `<CFTREE>` tag creates one of the most powerful and useful controls in ColdFusion—a branched tree control for the display of data. To add a tree control to a form, use opening and closing `<CFTREE>` tags, which control the tree's width, height, and other basic attributes.

Then, within the `<CFTREE>` tag pair, add one or more `<CFTREEITEM>` tags for each item you want to be displayed in the tree. Of course, as you would expect from ColdFusion, the `<CFTREEITEM>` tags easily can be generated from the results of a query.

Table 24.1 shows the attributes supported by the `<CFTREE>` tag, which creates the shell for the tree control itself. Table 24.2 explains the attributes for the `<CFTREEITEM>` tag, which populates the tree with actual information to show to the user.

Don't worry too much about how many attributes exist for these tags. As you will soon see, these tags are actually very easy to use.

Table 24.1 <CFTREE> Tag Attributes

ATTRIBUTE	DESCRIPTION
NAME	Required. A name for the tree control, similar to the NAME attribute for other form elements, such as <INPUT> and <CFSELECT>.
BORDER	Optional. Places a border around the tree. Default is Yes.
HSCROLL	Optional. Yes or No. Determines whether to show a horizontal scrollbar. Default is Yes.
VSCROLL	Optional. Yes or No. Determines whether to show a vertical scrollbar. Default is Yes.
REQUIRED	Optional. Yes or No. User must select an item in the tree control. Default is No.
MESSAGE	Optional. Message text to appear if REQUIRED="Yes" and the user attempts to submit the form without selecting an item in the tree first.
DELIMITER	Optional. The character used to separate elements in the treename.path form variable that is provided when a form that contains a tree is submitted. The default is \. See the section "Getting the Chosen Tree Item When the Form Is Submitted," later in this chapter.
COMPLETEPATH	Optional. Yes passes the root level of the tree in the treename.path form variable that is provided when a form that contains a tree is submitted. If omitted or No, the root level is not included. It is recommended that you set this value to Yes when using trees for data entry. See the section, "Getting the Chosen Tree Item When the Form Is Submitted," later in this chapter.
HIGHLIGHTHREF	Optional. Relevant only if the tree contains <CFTREEITEM> tags that specify an HREF attribute (see Table 24.2). Yes shows such items as underlined so they look like ordinary Web page hyperlinks; No gets rid of the underlining. Default is Yes.
APPENDKEY	Optional. Relevant only if the tree contains <CFTREEITEM> tags that specify an HREF attribute. Yes passes a URL variable called CFTREEITEMKEY when a user clicks such a tree item. The value of CFTREEITEMKEY indicates the VALUE of the selected <CFTREEITEM>. The default is Yes. In general, it is more straightforward to simply pass whatever URL parameters you want by adding them to the HREF attribute yourself, as shown in Listing 24.2, later in this chapter.

NOTE

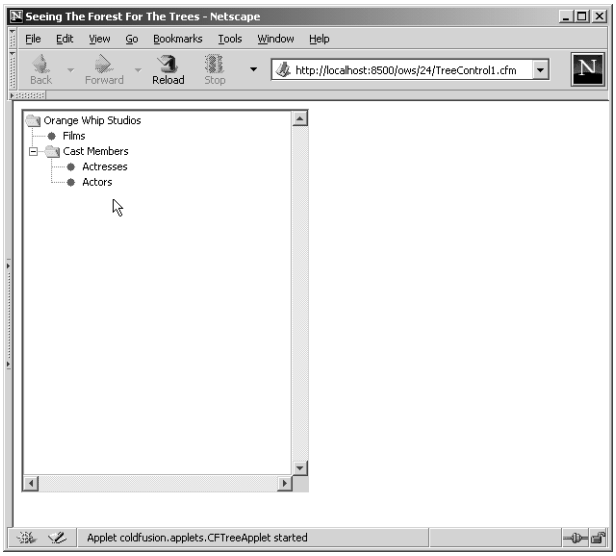
In addition, the <CFTREE> tag supports the look and feel attributes (such as LOOKANDFEEL and FONTSIZE) listed in Table 24.3. See the section "Controlling the Look and Feel," later in this chapter.

Table 24.2 `<CFTREEITEM>` Tag Attributes

ATTRIBUTE	DESCRIPTION
`DISPLAY`	Optional. The label to display to the user for the tree item. If you don't provide a value for `DISPLAY`, the `VALUE` attribute (see the following) is used as the default.
`VALUE`	Required. The underlying value (such as an ID number) that corresponds with the tree item. When the form is submitted, the value of the user-selected item will be available to the receiving template as the `FORM` variable specified by the `NAME` parameter of the `<CFTREE>` tag.
`PARENT`	Optional. The value of the tree item's parent, if any. Use this parameter to control which tree items are nested within other items. If you provide a value for `PARENT`, and the value corresponds with the `VALUE` of some other item in the tree, the new item will appear nested within the parent item, in a nested folder-like manner. If you don't provide a value for `PARENT` (or if the specified parent item doesn't exist in the tree), the new item becomes a root item in the tree (that is, not nested within any other items at all).
`EXPAND`	Optional. `Yes` or `No`. If `Yes` (the default), the tree item is expanded when the page first appears, revealing its children (if any). If `No`, the tree item is tree is closed when the page first appears; its children don't appear until the user expands the tree item by clicking it.
`IMG`	Optional. Image name or filename for the tree item. A number of images are supplied and can be specified using only the image name (no file extension): `folder`, `floppy`, `fixed`, `cd`, `document`, or `element`. The default image that appears when you don't specify an `IMG` value of your own varies depending on the `LOOKANDFEEL` attribute of the `<CFTREE>` tag. To specify your own custom image, specify the path in the same way that you would in an HTML `<IMG>` tag: `IMG="../images/filmicon.gif"`
`IMGOPEN`	Optional. The image to display when the tree item is expanded. You can use the same values or paths previously described for the `IMG` attribute.
`HREF`	Optional. URL to associate with the tree item or a query column for a tree that is populated from a query. If `HREF` is a query column, the `HREF` value is the value populated by the query. If `HREF` is not recognized as a query column, it is assumed that the `HREF` text is an actual `HREF`.
`TARGET`	Optional. Relevant only if the `HREF` attribute is specified. Target frame to activate if the user clicks the item. You can supply any of the values that you would normally supply to the `TARGET` attribute of an `<A>` or `<FORM>` tag (`_parent`, `_self`, `_top`, `_blank`, or a frame name you have defined yourself).
`QUERY`	Optional. Query name used to generate data for the tree item. If you provide a query name here, a tree item is generated for each row of the query. If you provide a `QUERY` attribute, the values you supply to the `VALUE`, `DISPLAY`, `HREF`, `TARGET`, and `IMG` attributes should be column names. In practice, using this `QUERY` attribute can get confusing rather quickly and often doesn't provide the flexibility required to create the trees you need for your application. It is generally recommended that you use a `<CFLOOP>` tag to iterate over the query, instead of using this attribute.
`QUERYASROOT`	Optional. Relevant only if you are providing a `QUERY` attribute. If `Yes`, the query itself appears as a item in the tree, using the query's name as the `DISPLAY` attribute. The default is `No`.

Listing 24.1 shows how to use <CFTREE> and <CFTREEITEM> to put together a basic tree. This tree contains five items which correspond to the five <CFTREEITEM> tags (Figure 24.1). The first item, labeled Orange Whip Studios, has no PARENT attribute and so appears at the root level of the tree. Because the Films and Cast Members items specify a PARENT attribute that corresponds to the VALUE of the first item (its value is root), they appear nested within the Orange Whip Studios item. Similarly, because the Actresses and Actors specify Cast as their PARENT, and because Cast is the VALUE of the Cast Members item, they appear nested within Cast Members when the form is displayed.

Figure 24.1
A simple <CFTREE> example.



Listing 24.1 TreeControl1.cfm—Building a Simple Tree Control with <CFTREE>

```
<!---
  Filename: TreeControl1.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Demonstrates use of the <CFTREE> tag
--->


<HTML>
<HEAD><TITLE>Seeing The Forest For The Trees</TITLE></HEAD>
<BODY>


<!--- Tree controls must appear between <CFFORM> tags --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="POST">

  <!--- Tree control --->
  <CFTREE
    NAME="NavTree"
    WIDTH="300"
    HEIGHT="400"
    BORDER="Yes"
```

Listing 24.1 (continued)

```
        APPENDKEY="No">

        <!--- Root tree item --->
        <CFTREEITEM
          VALUE="root"
          DISPLAY="Orange Whip Studios"
          EXPAND="Yes">
        <!--- Tree item for films --->
        <CFTREEITEM
          PARENT="root"
          VALUE="Films"
          DISPLAY="Films"
          EXPAND="No">
        <!--- Tree item for actors --->
        <CFTREEITEM
          PARENT="root"
          VALUE="Cast"
          DISPLAY="Cast Members"
          EXPAND="Yes">
        <!--- Tree item for actors --->
        <CFTREEITEM
          PARENT="Cast"
          VALUE="CastFemale"
          DISPLAY="Actresses"
          EXPAND="No">
        <!--- Tree item for actors --->
        <CFTREEITEM
          PARENT="Cast"
          VALUE="CastMale"
          DISPLAY="Actors"
          EXPAND="No">

      </CFTREE>
    </CFFORM>

    </BODY>
    </HTML>
```

As you can see in Figure 24.1, no really helpful information is being presented to the user yet. This example is here mainly to help you see how the `PARENT` and `VALUE` attributes work together to define the relationships between the items in a tree. The next listing adds more tree items that correspond to information in the database, which will make the tree a bit more interesting.

Using Queries to Generate Tree Items

Frequently, you will want to create trees that contain information from a database. For instance, you might use `<CFQUERY>` to run a query named `GetFilms` and then add a tree item for each film to a tree you're building.

There are two ways to add items to a tree based on the results of a query. The easiest, most flexible method is to use an ordinary `<CFLOOP>` tag that loops over the rows of your query. Within the `<CFLOOP>`, place a `<CFTREEITEM>` tag to include a tree item for each row returned by the query. Of course, you can refer to the query's columns in the `VALUE`, `DISPLAY`, and other attributes of the `<CFTREEITEM>` tag.

The second method is to use the QUERY attribute of the <CFTREEITEM> tag. This method is a bit more concise because it allows you to skip the <CFLOOP>, but it is generally more tedious to use in practice because any formatting, filtering, and so on must be done at the query level. The <CFLOOP> method gives you more flexibility and makes your code easier to understand; it is the one that is shown in the examples in this chapter.

Listing 24.2 expands on the tree created in Listing 24.1. This version includes the same static tree elements as the first version. It then adds tree items for each of Orange Whip Studio's quality films and actors. The films are organized by rating.

Listing 24.2 TreeControl2.cfm—Adding Queried Information to a <CFTREE>

```
<!---
  Filename: TreeControl2.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Demonstrates use of the <CFTREE> tag
--->

<HTML>
<HEAD><TITLE>Seeing The Forest For The Trees</TITLE></HEAD>
<BODY>

<!--- Get list of ratings from database --->
<CFQUERY NAME="GetRatings" DATASOURCE="ows">
  SELECT RatingID, Rating
  FROM FilmsRatings
  ORDER BY Rating
</CFQUERY>

<!--- Get information about films from database --->
<CFQUERY NAME="GetFilms" DATASOURCE="ows">
  SELECT FilmID, MovieTitle, RatingID
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- Fetch a listing of current actors from the database --->
<CFQUERY NAME="GetActors" DATASOURCE="ows">
  SELECT ActorID, NameFirst, NameLast, Gender
  FROM Actors
  ORDER BY NameLast, NameFirst
</CFQUERY>


<!--- Tree controls must appear between <CFFORM> tags --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="POST">

  <!--- Tree control --->
  <CFTREE
    NAME="NavTree"
    WIDTH="300"
    HEIGHT="400"
    BORDER="Yes"
    APPENDKEY="No">
```

Listing 24.2 (CONTINUED)

```
<!--- Root tree item --->
<CFTREEITEM
  VALUE="root"
  DISPLAY="Orange Whip Studios"
  EXPAND="Yes">
<!--- Tree item for films --->
<CFTREEITEM
  PARENT="root"
  VALUE="Films"
  DISPLAY="Films"
  EXPAND="No">

<!--- For each rating, generate a tree item --->
<!--- Specify Films as its parent, so it appears within films visually --->
<CFLOOP QUERY="GetRatings">
  <CFTREEITEM
    PARENT="Films"
    VALUE="Rating #RatingID#"
    DISPLAY="#Rating#">

  <!--- For each film with this rating, generate a tree item --->
  <CFLOOP QUERY="GetFilms">
    <CFIF GetFilms.RatingID EQ GetRatings.RatingiD[GetRatings.CurrentRow]>
      <!--- Generate actual tree item --->
      <CFTREEITEM
        PARENT="Rating #GetFilms.RatingID#"
        VALUE="#FilmID#"
        DISPLAY="#MovieTitle#"
        HREF="ShowFilm.cfm?FilmID=#FilmID#">
    </CFIF>
  </CFLOOP>

</CFLOOP>


<!--- Tree item for actors --->
<CFTREEITEM
  PARENT="root"
  VALUE="Cast"
  DISPLAY="Cast Members"
  EXPAND="Yes">
<!--- Tree item for actresses --->
<CFTREEITEM
  PARENT="Cast"
  VALUE="CastFemale"
  DISPLAY="Actresses"
  EXPAND="No">

<!--- Tree item for each actress --->
<CFLOOP QUERY="GetActors">
  <CFIF GetActors.Gender EQ "F">
    <!--- Generate actual tree item --->
    <CFTREEITEM
      PARENT="CastFemale"
```

Listing 24.2 (CONTINUED)

```
            VALUE="#ActorID#"
            DISPLAY="#NameFirst# #NameLast#"
            EXPAND="No"
            HREF="ShowActor.cfm?ActorID=#ActorID#">
        </CFIF>
      </CFLOOP>


      <!--- Tree item for actors --->
      <CFTREEITEM
        PARENT="Cast"
        VALUE="CastMale"
        DISPLAY="Actors"
        EXPAND="No">

      <!--- Tree item for each actor --->
      <CFLOOP QUERY="GetActors">
        <CFIF GetActors.Gender EQ "M">
          <!--- Generate actual tree item --->
          <CFTREEITEM
            PARENT="CastMale"
            VALUE="#ActorID#"
            DISPLAY="#NameFirst# #NameLast#"
            EXPAND="No"
            HREF="ShowActor.cfm?ActorID=#ActorID#">
        </CFIF>
      </CFLOOP>


    </CFTREE>
  </CFFORM>

  </BODY>
  </HTML>
```

Within each of the <CFLOOP> tags, a variable called ParentItem is set to determine which tree item the film or actor should appear within. For films, the ParentItem is determined by the film's rating; for actors, the ParentItem is determined by the actor's gender. The ParentItem variable is then fed to the PARENT attribute of <CFTREEITEM>, which causes the hierarchical relationship between the items to be expressed visually to the user.

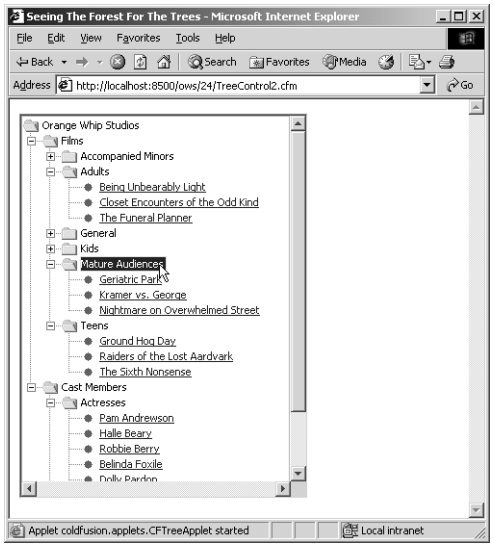
As you can see, tree controls are a great way to present a lot of information to the user in a familiar, organized manner (Figure 24.2). Because each of the generated <CFTREEITEM> tags for films and actors has HREF attributes, the user will be able to click the film or actor name to be taken to some type of detail page for that film or actor.

NOTE

Please note that the ShowActor.cfm and ShowFilm.cfm templates referred to in Listing 24.2 are not provided as examples for this chapter, so you will get Not Found errors if you actually try to click the underlined items in the tree (as shown in Figure 24.2). If you want, you could quickly put ShowActor.cfm and ShowFilm.cfm templates together that query the database based on the ActorID or FilmID parameters the <CFTREEITEM> tags pass in the URL.

Figure 24.2

This version of the <CFTREE> includes queried information about films and actors.



Using Trees for Data Entry

So far, the trees you have seen have all used the <CFTREEITEM> tag's HREF attribute to create links for the user to click. In other words, so far the tree control essentially has been used as a navigational construct—a structured container for URLs. As you already have seen in Listings 24.1 and 24.2, this can be a useful way to use the <CFTREE> tag.

You also can use the <CFTREE> tag to create a tree about data entry, rather than navigation. For instance, you might create a data-entry form for adding new expenses into the Expenses table, and you might want the user to identify a category for the new expense. If these categories are organized in a category/subcategory fashion, a natural and efficient way to display the categories is with a <CFTREE> control (look ahead at Figure 24.3 to see what we're talking about).

Working with Categories and Subcategories

A number of ways are available to provide a category/subcategory structure for things such as expenses, company departments, parts explosions, and so on. This section explains one of the most common and flexible approaches.

The basic idea is simple. You create a database table with columns called CatID, ParentCatID, and Description (or similar names). After the table has been created, fill it with categories and subcategories. For a top-level category, leave the ParentCatID column blank. For a subcategory, set the ParentCatID column to the parent category's CatID. This simple technique enables you to easily represent a hierarchical category structure in your database. You can have as many categories and subcategories as you need, and they can be nested within each other as deeply as you want. To move a category within some other category, you need only change its ParentCatID value.

There is no such table in the Orange Whip Studios sample database, so I will use a text file to hold the category and subcategory entries. The text file is shown in Listing 24.3.

NOTE

Normally, you would just create a `ExpenseCats` table in your database, rather than keeping the category information in a separate file. In fact, you might want to create such a table and adapt the following example to use that table instead. Just create the table with columns named `CatID`, `ParentCatID`, and `Description` and fill it with the rows of data shown in Listing 24.3.

Listing 24.3 ExpenseCats.txt—Sample Categories and Subcategories for Expenses

```
CatID,ParentCatID,Description
1,0,"One-Time Expenses"
2,1,"Props and Sets"
3,2,"Special Effects"
4,0,"Recurring Expenses"
5,4,"Food and Catering"
6,4,"Travel"
7,1,"Miscellaneous"
8,3,"Make-Up"
9,3,"Explosives"
10,1,"Properties"
```

This simple text file establishes two top-level categories, named `One-Time Expenses` and `Recurring Expenses` (they are top-level categories because their `ParentCatID` values have been left blank). Within the `One-Time Expenses` category are three subcategories—called `Props and Sets`, `Miscellaneous`, and `Properties` (they are subcategories of `One-Time Expenses` because their `ParentCatID` values match the `CatID` value for `One-Time Expenses`). Within `Props and Sets` is a subcategory named `Special Effects`; `Special Effects` has two subcategories named `Explosives` and `Make-Up`, and so on.

Displaying Categories and Subcategories in a Tree

Listing 24.4 shows how the category and subcategory information from Listing 24.3 can be displayed in a `<CFTREE>`. The `<CFTREE>` control is part of a data-entry form that requires the user to select an expense category from the tree, select a film from a drop-down list, and provide a description and expense amount (Figure 24.3).

Listing 24.4 ExpenseEntry.cfm—Displaying Categories and Subcategories in a Data-Entry Form

```
<!---
  Filename: ExpenseEntry.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Data entry interface with tree display of categories
--->


<!--- If the form is being submitted --->
<CFIF IsDefined("FORM.Description")>
  <!--- Database insersion code would go here. The chosen expense --->
  <!--- category from tree would be available as #FORM.ExpenseCat.node# --->
</CFIF>

<!--- Use <CFHTTP> to retrieve the ExpenseCats.txt file, parse its data as --->
```

Listing 24.4 (CONTINUED)

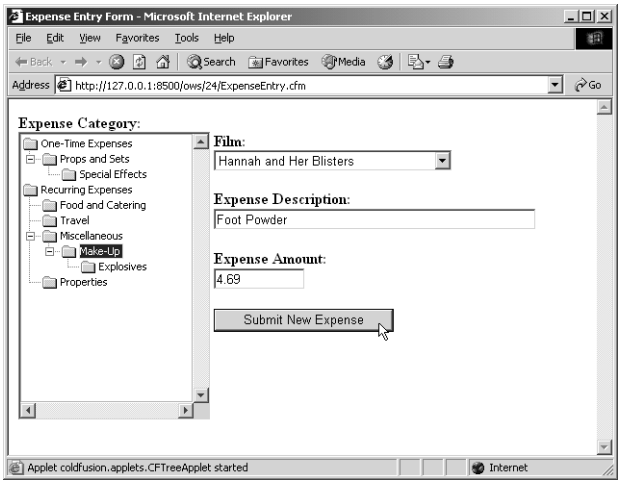
```
<!--- comma-separated text, and return it as a query object called GetCats --->
<CFHTTP
  URL="http://localhost:#CGI.SERVER_PORT#/ows/24/ExpenseCats.txt"
  NAME="GetCats"
  DELIMITER=","
  FIRSTROWASHEADERS="Yes">

<!--- Query Films.txt file for list of films --->
<CFQUERY NAME="GetFilms" DATASOURCE="ows">
  SELECT * FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<HTML>
<HEAD><TITLE>Expense Entry Form</TITLE></HEAD>
<BODY>

<!--- Tree controls must appear between <CFFORM> tags --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="POST">

  <!--- Tree control --->
  <B>Expense Category:</B><BR>
  <CFTREE
    NAME="ExpenseCat"
    WIDTH="200"
    HEIGHT="300"
    BORDER="Yes"
    COMPLETEPATH="Yes"
    REQUIRED="Yes"
    MESSAGE="Please pick a category from the tree."
    ALIGN="LEFT">

    <!--- For each category, generate a tree item --->
    <CFLOOP QUERY="GetCats">
      <CFTREEITEM
        PARENT="#GetCats.ParentCatID#"
        VALUE="#GetCats.CatID#"
        DISPLAY="#GetCats.Description#"
        IMG="folder">
    </CFLOOP>
  </CFTREE>

  <!--- Drop-down list of films --->
  <B>Film:</B><BR>
  <CFSELECT
    NAME="FilmID"
    QUERY="GetFilms"
    VALUE="FilmID"
    DISPLAY="MovieTitle"
    MESSAGE="Please choose a Film first."/><BR>

  <!--- Text entry field for expense description --->
  <P><B>Expense Description:</B><BR>
  <CFINPUT
    NAME="Description"
    SIZE="50"
    MAXLENGTH="200"
```

Listing 24.4 (CONTINUED)

```
        REQUIRED="Yes"
        MESSAGE="Please don't leave the Description blank."><BR>

      <!--- Text entry field for expense amount --->
      <P><B>Expense Amount:</B><BR>
      <CFINPUT
        NAME="ExpenseAmount"
        SIZE="10"
        MAXLENGTH="200"
        REQUIRED="Yes"
        VALIDATE="float"
        MESSAGE="You must fill in an amount first."><BR>

      <!--- Submit button for form --->
      <P><INPUT TYPE="Submit" VALUE="Submit New Expense">
    </CFFORM>

    </BODY>
    </HTML>
```

Figure 24.3

The <CFTREE> tag can be used as a data-entry control in your application's forms.



NOTE

The ALIGN="LEFT" attribute used in the <CFTREE> tag in this listing is explained in the section "Controlling the Look and Feel," later in this chapter.

The first thing this listing needs to do is to get access to the categories in the ExpenseCats.txt file (Listing 24.3). Because that file contains simple comma-separated text, the <CFHTTP> tag can be used to retrieve the file and parse the data in the file into a query object. After this <CFHTTP> tag executes, the information in the text file will be available as a query object called GetCats, which can be used just like any other query object (such as the results of a <CFQUERY> tag). Because of the FIRSTROWASHEADERS="Yes" attribute, the query object will contain columns named CatID, ParentCatID, and Description (from the first row of the text file). Please refer to Appendix B, "ColdFusion Tag Reference," for details about <CFHTTP>. If the category information was in a table in your database, you could just use an ordinary <CFQUERY> instead of <CFHTTP>.

NOTE

The `CGI.SERVER_NAME` and `CGI.SERVER_PORT` variables are used in the `URL` attribute of the `<CFHTTP>` tag so that the listing will continue to work regardless of what server you place this listing on, or whether ColdFusion MX has been installed in stand-alone mode. If you want, you could just hardcode the `localhost:8500` part of the URL (or whatever is appropriate for your server) instead. For details about these `CGI` variables, please refer to Appendix D, "Special ColdFusion Variables and Result Codes".

Next, a normal query named `GetFilms` also is used to get a list of current films from the database, and a data entry form is created using the `<CFFORM>` tag. The form's `ACTION` is set to `#CGI.SCRIPT_NAME#`, which means the `ExpenseEntry.cfm` file will be called again when the user submits the form. See Appendix D, "Special ColdFusion Variables and Result Codes," for more information about this `CGI` variable.

The most important control within the form is the `<CFTREE>` tag, which allows the user to pick the appropriate expense category from the tree. The `REQUIRED` attribute is used to ensure that the user has indeed selected a category in the tree before he can submit the form; if not, he sees the message supplied to the `MESSAGE` attribute. Note that `REQUIRED` and `MESSAGE` work just like the equivalent attributes for the `<CFINPUT>` tag; see Chapter 12 for more information about this type of data validation.

Within the `<CFTREE>` tag pair, a `<CFLOOP>` is used to add a tree item for each row in the `GetCats` query. Because the `PARENT` and `VALUE` attributes are supplied with the category's `CatID` and `ParentCatID` values, the tree will visually reflect the category/subcategory structure implied by the data in the `ExpenseCats.txt` file (refer to Listing 24.3).

The rest of the form is straightforward and will be familiar to you if you have read Chapters 12 and 13. If you have not, you might want to take a glance at those chapters now if the use of `<CFINPUT>` and `<CFSELECT>` is unfamiliar to you.

Getting the Chosen Tree Item When the Form Is Submitted

After a form with a `<CFTREE>` on it is submitted, ColdFusion makes the user's choice available to you in the `FORM` scope, similar to other types of form fields. However, instead of providing you with just a single value, ColdFusion provides you with two values for each tree in the form. The first value is called `Node` and indicates the `VALUE` of the `<CFTREEITEM>` that the user selected before submitting the form. The second value is called `Path` and indicates the parent tree items of the selected item, if any.

Both values are named after the `NAME` of the `<CFTREE>` tag, using dot notation. For instance, after the form in Listing 24.4 is submitted, there will be a variable called `FORM.ExpenseCat.Node`, which would hold a value of `9` if the user has selected the Explosives item in the tree. There will also be a variable called `FORM.ExpenseCat.Path`, which will be set to `1\2\3\9` to indicate that the selected category was `9`, which was nested within category `3` (Special Effects), which was in category `2` (Props and Sets), which was in category `1` (One-Time Expenses).

Listing 24.4 doesn't actually insert a record into the `Expenses` table when the form is submitted because the `Expenses` table doesn't currently have a column to hold the expense category number. If you were to add such a column (called `ExpenseCatID`, for instance), you could populate the column

when the form was submitted by using a query similar to the following. You would place this between the <CFIF> tags at the top of the template:

```
<CFQUERY DATASOURCE="ows">
  INSERT INTO Expenses(FilmID, ExpenseCatID, ExpenseAmount, Description)
  VALUES (#FORM.FilmID#, #FORM.ExpenseCat.Node#,
          #FORM.ExpenseAmount#, '#FORM.Description#')
</CFQUERY>
```

Note that the FORM.ExpenseCat.Node variable is used to supply the value for the fictitious ExpenseCatID column.

Controlling the Look and Feel

ColdFusion enables you to control the look and feel of the <CFTREE> and other Java-based controls discussed in this chapter (<CFGRID>, <CFSLIDER>, and <CFTEXTINPUT>). Table 24.3 lists the additional attributes you can add to these tags to tweak the way they appear in your application's pages. If you want, you can experiment a bit with these attributes by adding them to the <CFTREE> tag in Listing 24.4.

Table 24.3 Common Attributes for Controlling the Look of the Java-Based <CFFORM> Controls

ATTRIBUTE	DESCRIPTION
LOOKANDFEEL	Optional. Setting that describes the general look and feel of the control, such as the way its borders are drawn and the general three-dimensional effect it has. Valid values for this attribute are Windows, Metal, and Motif. The default is the look and feel appropriate for the user's machine.
FONT	Optional. Font name for text.
FONTSIZE	Optional. Font size for text.
BOLD	Optional. Enter Yes for boldface text and No for regular text. This has the same effect as <B></B> in HTML. The default is No.
ITALIC	Optional. Enter Yes for italicized text and No for normal text. This has the same effect as <I></I> in HTML. Default is No.
ALIGN	Optional. Alignment value for the CFFORM element. Valid entries are top, middle, left, right, bottom, baseline, texttop, absbottom, and absmiddle. Similar to the ALIGN attribute of the HTML <IMG> tag.
BORDER	Optional. Whether to show a border around the control. Yes or No.
VSPACE	Optional. Amount of vertical space to reserve as an invisible margin (gutter) above and below the control. You can use VSPACE and HSPACE together to ensure the control doesn't appear to be too close to other elements on the page. Similar to the VSPACE attribute of the HTML <IMG> tag.
HSPACE	Optional. Amount of horizontal space to reserve as an invisible margin (gutter) to the left and right of the control. You can use VSPACE and HSPACE together to ensure the control doesn't appear to be too close to other elements on the page. Similar to the HSPACE attribute of the HTML <IMG> tag.
WIDTH	Optional. The control's width, in pixels.
HEIGHT	Optional. The control's height, in pixels.

If the Control Does Not Load

Normally, you should be able to just go ahead and start using the <CFTREE>, <CFGRID>, and <CFSLIDER> controls discussed in this chapter. However, if your ColdFusion templates are being served on a virtual Web server (that is, a single machine that hosts a number of Web sites, each with its own document root), or if certain directories have been deleted, you might need to take a few extra steps before the controls will work.

Basically, the user's Web browser must be capable of accessing a file called cfapplets.jar, at the relative location /CFIDE/classes/cfapplets.jar. After a typical ColdFusion installation, this should already be the case (there will be a folder within your Web server's document root called CFIDE, which will contain a folder called classes, which will contain the cfapplets.jar file). If the Java-based controls do not appear to load properly—especially if a class-not-found error is displayed in the browser's status line—try to visit the following URL in your browser (substituting www.yourcompany.com with the hostname for your ColdFusion server):

```
http://www.yourcompany.com/CFIDE/classes/cfapplets.jar
```

If you get a Not Found, Forbidden, 404, or some similar error message, then the browser is incapable of accessing the cfapplets.jar file for one reason or another. There are generally two ways to solve this problem:

- Create a folder called CFIDE within your document root, and copy the classes folder (including the cfapplets.jar file) into it. You would copy the file from the Web server's default document root, if you have access to it, or from some other ColdFusion server (perhaps your local machine). If the folder already exists, ensure that the cfapplets.jar file is still there and doesn't have some kind of password (or other type of access control) protecting it.
- Or, using your Web server software's configuration tools, create a virtual folder or virtual directory called CFIDE that points to the CFIDE folder in the Web server's default document root. This assumes that the folder hasn't been deleted. The steps you take to accomplish this will vary depending on the Web server software you are using.

Using <CFGRID>

In early versions of ColdFusion, the <CFGRID> tag was simply a method of presenting spreadsheet-style grids of information to your users. Over the years, <CFGRID> has become a capable data-entry tool as well (meaning that users can edit the information in the grid and submit it to the server for processing). The current version enables the adding of new data rows, deleting of rows, sorting of columns, and instant updating of all data in the grid. It even allows you to show check boxes and drop-down lists within the grid and provides a simple way to color-code individual cells on-the-fly. In short, the tag provides a nice, clean way of exposing spreadsheet-type data on a Web page.

As with the other <CFFORM> tags, you really have to build some sample applications to see what this tag looks like and what it does. <CFGRID> has more than 30 attributes you can use to customize its

appearance and behavior. Any attributes set in the <CFGRID> tag affect the entire grid. You can also specify options for specific rows or columns by passing attributes to <CFGRIDCOLUMN> and <CFGRIDROW>. Options passed to <CFGRIDCOLUMN> and <CFGRIDROW> override any options set at the <CFGRID> level.

Table 24.4 is an abbreviated list of the attributes supported by <CFGRID>. I've kept this list short to make it easier for you to concentrate on the most important attributes. The complete list is provided in Appendix B, "ColdFusion Tag Reference."

Table 24.4 An Abbreviated List of <CFGRID> Tag Attributes

ATTRIBUTE	DESCRIPTION
NAME	Optional. A name for the grid, which should be unique for the form.
QUERY	Optional. The name of the query to be displayed in the grid. Required unless you use the <CFGRIDROW> tag (see Table 24.6, later in this chapter).
MAXROWS	Optional. Maximum number of query rows to display; same as the MAXROWS attribute for the <CFOUTPUT> tag.
COLHEADERS	Optional. Yes to display column headers at the top of the grid; No to hide column headers. Default is Yes. You have further control over the column headers via COLHEADERALIGN, COLHEADERFONT, COLHEADERFONTSIZE, COLHEADERTEXTCOLOR, COLHEADERITALIC, and COLHEADERBOLD (see Appendix B for details).
ROWHEADERS	Optional. Yes to display row headers at the left edge of the grid; No to hide row headers. Default is Yes. You have further control over the look of the row headers via ROWHEADERALIGN, ROWHEADERFONT, ROWHEADERFONTSIZE, ROWHEADERITALIC, ROWHEADERBOLD, ROWHEADERWIDTH, ROWHEIGHT, and ROWHEADERTEXTCOLOR.
SELECTMODE	Optional. Controls how users can select cells, if at all, and enables editing. Valid entries are BROWSE (no selecting or editing), EDIT (user can edit the data in the grid), SINGLE (user can select a single cell, but can't edit it), ROW (user can select only whole rows at a time), and COLUMN (user can select only whole columns at a time). The default is BROWSE.
INSERT	Optional. Relevant only if SELECTMODE="EDIT". If set to Yes, the user is provided with a button to add new records to the grid. The label shown on the button can be controlled with INSERTBUTTON (see Appendix A). Default is No.
DELETE	Optional. Relevant only if SELECTMODE="EDIT". If set to Yes, the user is provided with a button to delete records from the grid. The label shown on the button can be controlled with DELETEBUTTON (see Appendix B). Default is No.

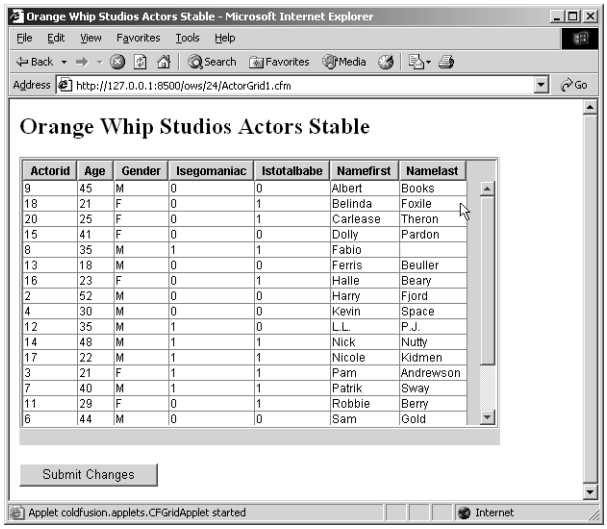
NOTE

The <CFGRID> tag also supports most of the look and feel attributes listed in Table 24.3, earlier in this chapter.

The <CFGRID> tag can be used alone or in conjunction with <CFGRIDCOLUMN> and <CFGRIDROW> tags. When used alone, the grid simply displays all the data returned by the query specified in its QUERY attribute (Figure 24.4). All columns are displayed, and you don't have much control over the individual columns (in terms of width, header, and so on). But it's ridiculously easy. Listing 24.5 shows how to use <CFGRID> in this way.

Figure 24.4

When used without <CFGRIDCOLUMN>, the <CFGRID> tag produces a default grid based on the column names in the query.



Listing 24.5 ActorGrid1.cfm—A Simple <CFGRID> Example

```
<!---
  Filename: ActorGrid1.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Demonstrates use of the <CFGRID> tag
--->


<!--- Get list of Actors from database --->
<CFQUERY NAME="GetActors" DATASOURCE="ows">
  SELECT ActorID, NameFirst, NameLast, Age, Gender, IsEgomaniac, IsTotalBabe
  FROM Actors
  ORDER BY NameFirst, NameLast
</CFQUERY>


<HTML>
<HEAD><TITLE>Orange Whip Studios Actors Stable</TITLE></HEAD>
<BODY>
<H2>Orange Whip Studios Actors Stable</H2>


<!--- Data entry form --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="Post">

  <!--- Grid control for expenses --->
  <CFGRID
    NAME="ActorGrid"
    QUERY="GetActors"
    WIDTH="500"
    HEIGHT="300"
    ROWHEADERS="No"
    COLHEADERBOLD="Yes"
```

Listing 24.5 (CONTINUED)

```
        COLHEADERALIGN="CENTER">
      </CFGRID>

      <P><INPUT TYPE="Submit" VALUE="Submit Changes">
    </CFFORM>

    </BODY>
    </HTML>
```

This example uses the minimum code necessary for a grid. No <CFGRIDCOLUMN> tag is needed because the database fields were used as the default column names. In general, the resulting grid will be too clunky to present to end users, but it will probably be good enough for putting together quick administrative pages for you (or your team's) own use.

NOTE

When you use <CFGRID> in this way, the order of the columns is determined by the alphabetical order of the column names. To control the order, use <CFGRIDCOLUMN> as discussed in the next section.

Adding <CFGRIDCOLUMN> Tags

To produce more sophisticated grids, you must add <CFGRIDCOLUMN> tags between your opening and closing <CFGRID> tags. This enables you to set display options for each individual column, such as header labels, column widths, alignments, and so on.

Table 24.5 is an abbreviated list of the attributes supported by <CFGRIDCOLUMN>. We've shortened the list here to make it easier for you to concentrate on the most important attributes. The complete list is provided in Appendix B.

Table 24.5 An Abbreviated List of <CFGRIDCOLUMN> Attributes

ATTRIBUTE	DESCRIPTION
NAME	Required. A name for the column; this should be the name of the query column you want to display in this column of the grid.
HEADER	Optional. Text to display in the column's header; the NAME attribute's text is used if this is omitted.
WIDTH	Optional. The width of the column, in pixels.
DISPLAY	Optional. Yes or No. If set to No, the column is hidden from view but is still submitted to the server when the form is submitted. In general, you use DISPLAY="No" for a table's ID number column. Similar conceptually to an <INPUT> form field of TYPE="Hidden". Default is Yes.
TYPE	Optional. Controls how the column can be sorted by the user by clicking the column header. If numeric, the user can sort the column numerically. If text or textnocase, the column is sorted alphabetically (text_nocase is generally more user friendly). You can also set TYPE="boolean" to make check boxes appear in the column instead of text values. This is a great way to enable users to turn a database table's Yes/No or Bit type columns on and off. You also can set TYPE="Image" to display images in the grid; see Appendix B or the ColdFusion documentation for details.

Table 24.5 (CONTINUED)

ATTRIBUTE	DESCRIPTION
DATAALIGN	Optional. The alignment for the data shown in the column. Valid entries are Left, Right, and Center. Defaults to Left.
NUMBERFORMAT	Optional. A formatting mask for displaying numbers. You can use the same mask characters supported by the NumberFormat() function (see Appendix C).
TEXTCOLOR	Optional. The color of the text in the column's cells, as a named color or hexadecimal RBG value. The color can change conditionally in real time, based on a formula you provide; see the section "Conditional Color Coding" for more information.
BGCOLOR	Optional. The background color for the column's cells, as a named color or hexadecimal RBG value. The color can change conditionally in real time, based on a formula you provide; see the section "Conditional Color Coding" for more information.
VALUES	Optional. If you provide a VALUES attribute and the column is editable (see SELECTMODE attribute in Table 24.4), the column becomes a drop-down list. The VALUES attribute is assumed to be a ColdFusion list, in which each list value creates a new choice in the drop-down list. Similar conceptually to the VALUE attribute of the <CFSELECT> tag (see Chapter 13).
VALUESDISPLAY	Optional. Relevant only if you provide a VALUES attribute. A list of values to show in the drop-down list. The VALUESDISPLAY attribute should also be a ColdFusion list, in which each list value is the user-friendly version of the corresponding list value in VALUES. Similar conceptually to the DISPLAY attribute of the <CFSELECT> tag.
VALUESDELIMITER	Optional. Relevant only if you provide a VALUES attribute. The delimiter character to use for the lists supplied to the VALUES and VALUESDISPLAY attributes. If the individual VALUESDISPLAY values might contain commas, you should change this to a delimiter that will not show up in your data, such as a semicolon or pipe character. See Chapter 8, "Using ColdFusion," for more information about ColdFusion lists and delimiters.

Listing 24.6 builds on the code from Listing 24.5, this time using <CFGRIDCOLUMN> tags to set individual attributes for each column that should be displayed in the grid. The ActorID column is hidden from view, and the widths and alignments and number masks of the other columns have been tweaked to make the grid more visually pleasing (Figure 24.5).

Listing 24.6 ActorGrid2.cfm—Using <CFGRIDCOLUMN> to Control the Display of Individual Columns

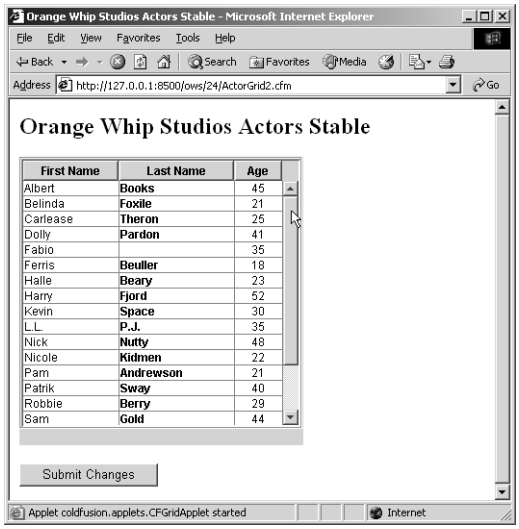
```
<!---
  Filename: ActorGrid2.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Demonstrates use of the <CFGRID> tag
--->
```

Listing 24.6 (CONTINUED)

```
<!--- Get list of Actors from database --->
<CFQUERY NAME="GetActors" DATASOURCE="ows">
  SELECT * FROM Actors
  ORDER BY NameFirst, NameLast
</CFQUERY>


<HTML>
<HEAD><TITLE>Orange Whip Studios Actors Stable</TITLE></HEAD>
<BODY>
<H2>Orange Whip Studios Actors Stable</H2>


<!--- Data entry form --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="Post">

  <!--- Grid control for expenses --->
  <CFGRID
    NAME="ActorGrid"
    QUERY="GetActors"
    WIDTH="295"
    HEIGHT="300"
    ROWHEADERS="No"
    COLHEADERBOLD="Yes"
    COLHEADERALIGN="CENTER"
    SELECTMODE="ROW">
    <!--- Hidden column for ActorID - does not actually get displayed --->
    <CFGRIDCOLUMN
      NAME="ActorID"
      DISPLAY="No">
    <!--- Column for actor's first name --->
    <CFGRIDCOLUMN
      NAME="NameFirst"
      HEADER="First Name"
      WIDTH="100">
    <!--- Column for actor's last name --->
    <CFGRIDCOLUMN
      NAME="NameLast"
      HEADER="Last Name"
      WIDTH="120"
      BOLD="Yes">
    <!--- Column for actor's age--->
    <CFGRIDCOLUMN
      NAME="Age"
      HEADER="Age"
      DATAALIGN="Center"
      WIDTH="50"
      NUMBERFORMAT="999.">
  </CFGRID>

  <P><INPUT TYPE="Submit" VALUE="Submit Changes">
</CFFORM>

</BODY>
</HTML>
```

Figure 24.5

The <CFGRIDCOLUMN> tag gives you additional control over the display of your grids.



Making Editable Grids

To make a grid editable, set the SELECTMODE of the <CFGRID> column to EDIT. The user will then be able to edit the data in the grid, almost as if she were using a spreadsheet program such as Microsoft Excel. If you enable the INSERT and DELETE options for the <CFGRID> (see Table 24.5), she will even be able to insert new records and delete existing ones from the grid.

TIP

If you want the user to be able to edit only certain columns, set the SELECT attribute of the <CFGRIDCOLUMN> tag (see Table 24.5).

Using the <CFGRIDUPDATE> Tag

As the user edits the values in the grid, she is changing the values only on her local machine (that is, the Java-based grid applet generated by <CFGRID> is not connected in real time to the database on the ColdFusion server). After a user has edited the grid to her satisfaction, she should have a way to commit her edits to the underlying database table. ColdFusion makes this easy to accomplish by providing the <CFGRIDUPDATE> tag, which takes care of updating the database accordingly when your users submit forms that contain editable grids.

The <CFGRIDUPDATE> tag is conceptually similar to the <CFUPDATE> tag you learned about in Chapter 13. In general, you need only provide the tag with the GRID, DATASOURCE, and TABLENAME attributes; the tag will take care of the rest. Any rows the user has added to the grid will be inserted into the specified database table, any rows the user has removed from the grid will be deleted from the table, and any existing cells the user has edited will be updated accordingly.

Table 24.6 is an abbreviated list of the attributes supported by <CFGRIDUPDATE>. I've shortened the list here to make it easier for you to concentrate on the most important attributes. The other attributes are explained in Appendix B.

Table 24.6 An Abbreviated List of <CFGRIDUPDATE> Attributes

NAME	STATUS	DESCRIPTION
GRID	Required.	The NAME of the <CFGRID> form element being submitted.
DATASOURCE	Required.	The name of the data source for the update action. Presumably the same DATASOURCE name you provided to the <CFGRID> tag.
TABLENAME	Required.	The name of the table you want to update. Presumably the same table from which you selected records to populate the <CFGRID>.
KEYONLY	Optional. Yes or No.	Yes specifies that in the update action, the WHERE criteria is confined to the key values. No specifies that in addition to the key values, the original values of any changed fields are included in the WHERE criteria. The default is Yes. In other words, if this attribute is set to No, records that have been changed by some other user since the page was loaded will not be changed again when the grid is submitted. Only those database rows that are unchanged since the grid was originally displayed will be affected by the <CFGRIDUPDATE> action. This is a simple but very effective way of ensuring that several users who are editing the same table at the same time do not overwrite each other's changes.

NOTE

The <CFGRIDUPDATE> tag also supports DATASOURCE, USERNAME, PASSWORD, TABLEOWNER, and TABLEQUALIFIER attributes, which work the same way as the corresponding attributes for the <CFUPDATE> tag discussed in Chapter 13. See Appendix B for details about these attributes.

Committing the User's Edits to the Database

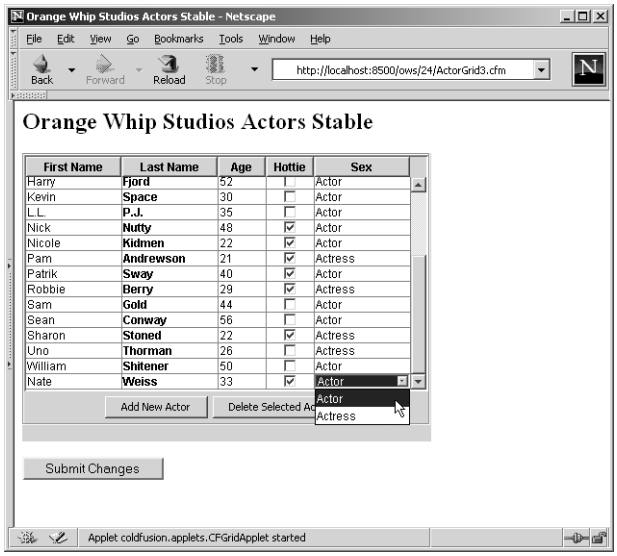
Listing 24.7 takes the previous version of the template (refer to Listing 24.6) and makes it editable. After the user makes whatever edits she desires and submits the form, the <CFIF> test at the top of the template evaluates to True, which causes the <CFGRIDUPDATE> tag to execute. The user can now edit the data in the grid, add new records, delete existing records, and submit the changes to the server (Figure 24.6).

TIP

Always consider setting the KEYONLY attributes to No (see Table 24.6) to prevent multiple users from accidentally overwriting each other's edits to the database.

Figure 24.6

Editable grids provide an easy way to enable users to edit multiple rows of data.



Listing 24.7 ActorGrid3.cfm—An Editable Grid

```
<!---
  Filename: ActorGrid3.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Demonstrates use of the <CFGRID> tag
--->

<!--- If form is being submitted, commit changes to the database --->
<CFIF CGI.REQUEST_METHOD is "Post">
  <CFGRIDUPDATE
    DATASOURCE="ows"
    TABLENAME="Actors"
    GRID="ActorGrid"
    KEYONLY="No">
</CFIF>

<!--- Get list of Actors from database --->
<CFQUERY NAME="GetActors" DATASOURCE="ows">
  SELECT * FROM Actors
  ORDER BY NameFirst, NameLast
</CFQUERY>


<HTML>
<HEAD><TITLE>Orange Whip Studios Actors Stable</TITLE></HEAD>
<BODY>
<H2>Orange Whip Studios Actors Stable</H2>


<!--- Data entry form --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="Post">
```

Listing 24.7 (CONTINUED)

```
    <!--- Grid control for expenses --->
    <CFGRID
      NAME="ActorGrid"
      QUERY="GetActors"
      WIDTH="425"
      HEIGHT="300"
      ROWHEADERS="No"
      COLHEADERBOLD="Yes"
      COLHEADERALIGN="CENTER"
      SELECTMODE="EDIT"
      INSERT="Yes"
      DELETE="Yes"
      INSERTBUTTON="Add New Actor"
      DELETEBUTTON="Delete Selected Actor">
      <!--- Hidden column for ActorID - does not actually get displayed --->
      <CFGRIDCOLUMN
        NAME="ActorID"
        DISPLAY="No">
      <!--- Column for actor's first name --->
      <CFGRIDCOLUMN
        NAME="NameFirst"
        HEADER="First Name"
        WIDTH="100">
      <!--- Column for actor's last name --->
      <CFGRIDCOLUMN
        NAME="NameLast"
        HEADER="Last Name"
        WIDTH="100"
        BOLD="Yes">
      <!--- Column for actor's age--->
      <CFGRIDCOLUMN
        NAME="Age"
        HEADER="Age"
        WIDTH="50"
        NUMBERFORMAT="999.">
      <!--- Column for babe factor --->
      <CFGRIDCOLUMN
        NAME="IsTotalBabe"
        HEADER="Hottie"
        WIDTH="50"
        TYPE="boolean">
      <!--- Column for gender --->
      <CFGRIDCOLUMN
        NAME="Gender"
        HEADER="Sex"
        WIDTH="100"
        VALUES="M,F"
        VALUESDISPLAY="Actor,Actress">
    </CFGRID>

    <P><INPUT TYPE="Submit" VALUE="Submit Changes">
  </CFFORM>

  </BODY>
  </HTML>
```

NOTE

This template uses the automatic `CGI.REQUEST_METHOD` variable to determine whether the form is currently being submitted. If the `REQUEST_METHOD` is `Post`, that is an indication that the form (which has a `METHOD` attribute of `Post`) is being submitted. If the page is simply being viewed normally (not as part of a form submission), the `REQUEST_METHOD` is `Get` instead. See Appendix D for details.

Processing Grid Edits Without <CFGRIDUPDATE>

As Listing 24.7 shows, the <CFGRIDUPDATE> tag makes updating a database table after a user makes changes to an editable grid easy. It handles all the appropriate inserts, updates, and deletes for you, all with just one line of code. In most cases, the <CFGRIDUPDATE> tag provides everything you need.

In some cases, however, you might want to handle each edit operation on your own. For instance, perhaps you want the user to be able to delete records from the `Actors` table, but only those actors who have not actually appeared in any films. ColdFusion enables you to handle this type of integrity check or other special processing by exposing a number of special arrays you can inspect to find out which edits the user made to the grid. Using the arrays, you can handle each edit operation separately, even ignoring some of the edits when appropriate.

Table 24.7 shows the special arrays ColdFusion makes available when a <CFGRID> is submitted. Normally, these array variables are used internally by the <CFGRIDUPDATE> tag to update the database automatically, but you can use them to create customized update handling behavior.

Each of the variables listed in Table 24.7 is a one-dimensional array. Each of the arrays always has the same number of elements in it, one for each grid row that was changed (edited, inserted, or deleted). Therefore, you can use the `ArrayLen()` function to find out how many edits were made to the grid.

Table 24.7 Special Arrays Available After a **<CFGRID>** Submission

FORM ATTRIBUTE	DESCRIPTION
`gridname.RowStatus.Action[index]`	The type of edit made to the row. Will be `U`, `I`, or `D` to indicate that the row was updated, inserted, or deleted, respectively.
`gridname.colname[index]`	The new value of the column indicated by `colname`.
`gridname.original.colname[index]`	The original value of the column indicated by `colname`.

NOTE

It is important to understand that these arrays will *not* contain an entry for each row in the grid; they will contain only entries for each row in the grid that has been changed (edited, inserted, or deleted).

Listing 24.8 demonstrates how to use these special arrays. As you can see, using these arrays instead of <CFGRIDUPDATE> generally requires a lot of additional code. Of course, much of this code could be eliminated if your grid did not allow inserts or deletes (see Table 24.4).

Listing 24.8 ActorGrid3b.cfm—Handling Each Update on Your Own

```
<!---
  Filename: ActorGrid3b.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Demonstrates use of the <CFGRID> tag
--->


<HTML>
<HEAD><TITLE>Orange Whip Studios Actors Stable</TITLE></HEAD>
<BODY>
<H2>Orange Whip Studios Actors Stable</H2>

<!--- If form is being submitted, commit changes to the database --->
<CFIF CGI.REQUEST_METHOD is "Post">
  <!--- The RowStatus.Action array contains a value for each change --->
  <CFSET NumChanges = ArrayLen(FORM.ActorGrid.RowStatus.Action)>

  <!--- For each change made to the grid... --->
  <CFLOOP FROM="1" TO="#NumChanges#" INDEX="i">
    <!--- What type of change was made? --->
    <CFSET ThisAction = FORM.ActorGrid.RowStatus.Action[i]>

    <!--- Depending on the type of change (update, insert, or delete) --->
    <CFSWITCH EXPRESSION="#ThisAction#">
      <!--- Handle updates (if existing values were edited in grid) --->
      <CFCASE VALUE="U">
        <CFQUERY DATASOURCE="ows">
          UPDATE Actors SET
            NameFirst = '#FORM.ActorGrid.NameFirst[i]#',
            NameLast = '#FORM.ActorGrid.NameLast[i]#',
            Age = #FORM.ActorGrid.Age[i]#,
            IsTotalBabe = #FORM.ActorGrid.IsTotalBabe[i]#,
            Gender = '#FORM.ActorGrid.Gender[i]#'
          WHERE
            ActorID = #FORM.ActorGrid.original.ActorID[i]#
        </CFQUERY>
      </CFCASE>
      <!--- Handle inserts (if record was added to the grid) --->
      <CFCASE VALUE="I">
        <!--- The boss says we're only hiring beautiful people from now on --->
        <CFIF FORM.ActorGrid.IsTotalBabe[i] EQ 1>
          <CFQUERY DATASOURCE="ows">
            INSERT INTO Actors (
              NameFirst,
              NameLast,
              NameFirstReal,
              NameLastReal,
              Age,
              IsTotalBabe,
              Gender
            ) VALUES (
              '#FORM.ActorGrid.NameFirst[i]#',
              '#FORM.ActorGrid.NameLast[i]#',
              '#FORM.ActorGrid.NameFirst[i]#',
              '#FORM.ActorGrid.NameLast[i]#',
```

Listing 24.8 (CONTINUED)

```
                #FORM.ActorGrid.Age[i]#,
                #FORM.ActorGrid.IsTotalBabe[i]#,
                '#FORM.ActorGrid.Gender[i]#'
              )
            </CFQUERY>
          <CFELSE>
           <CFOUTPUT>
            <P>Sorry, #FORM.ActorGrid.NameFirst[i]# #FORM.ActorGrid.NameLast[i]#
            cannot be added because it's company policy to only hire Hotties.<BR>
           </CFOUTPUT>
          </CFIF>
        </CFCASE>
        <!--- Handle deletes (if existing row was deleted from grid) --->
        <CFCASE VALUE="D">
          <!--- Find out if this actor is in any films --->
          <CFQUERY DATASOURCE="ows" NAME="GetActorFilms">
            SELECT FilmID FROM FilmsActors
            WHERE ActorID = #FORM.ActorGrid.original.ActorID[i]#
          </CFQUERY>
          <!--- Only allow a delete if the actor is not in any films --->
          <CFIF GetActorFilms.RecordCount EQ 0>
            <CFQUERY DATASOURCE="ows">
              DELETE FROM Actors
              WHERE ActorID = #FORM.ActorGrid.original.ActorID[i]#
            </CFQUERY>
          <CFELSE>
            <CFOUTPUT>
              <P>Sorry, #FORM.ActorGrid.original.NameFirst[i]#
              #FORM.ActorGrid.original.NameLast[i]#
              cannot be deleted because that actor has been in
              #GetActorFilms.RecordCount# of our films already.<BR>
            </CFOUTPUT>
          </CFIF>
        </CFCASE>
      </CFSWITCH>
    </CFLOOP>
  </CFIF>

  <!--- Get list of Actors from database --->
  <CFQUERY NAME="GetActors" DATASOURCE="ows">
    SELECT * FROM Actors
    ORDER BY NameFirst, NameLast
  </CFQUERY>


  <!--- Data entry form --->
  <CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="Post">

    <!--- Grid control for expenses --->
    <CFGRID
      NAME="ActorGrid"
      QUERY="GetActors"
      WIDTH="425"
      HEIGHT="300"
      ROWHEADERS="No"
      COLHEADERBOLD="Yes"
      COLHEADERALIGN="CENTER"
```

Listing 24.8 (CONTINUED)

```
        SELECTMODE="EDIT"
        INSERT="Yes"
        DELETE="Yes"
        INSERTBUTTON="Add New Actor"
        DELETEBUTTON="Delete Selected Actor">
        <!--- Hidden column for ActorID - does not actually get displayed --->
        <CFGRIDCOLUMN
          NAME="ActorID"
          DISPLAY="No">
        <!--- Column for actor's first name --->
        <CFGRIDCOLUMN
          NAME="NameFirst"
          HEADER="First Name"
          WIDTH="100">
        <!--- Column for actor's last name --->
        <CFGRIDCOLUMN
          NAME="NameLast"
          HEADER="Last Name"
          WIDTH="100"
          BOLD="Yes">
        <!--- Column for actor's age--->
        <CFGRIDCOLUMN
          NAME="Age"
          HEADER="Age"
          WIDTH="50"
          NUMBERFORMAT="999.">
        <!--- Column for babe factor --->
        <CFGRIDCOLUMN
          NAME="IsTotalBabe"
          HEADER="Hottie"
          WIDTH="50"
          TYPE="boolean">
        <!--- Column for gender --->
        <CFGRIDCOLUMN
          NAME="Gender"
          HEADER="Sex"
          WIDTH="100"
          VALUES="M,F"
          VALUESDISPLAY="Actor,Actress">
      </CFGRID>

      <P><INPUT TYPE="Submit" VALUE="Submit Changes">
    </CFFORM>

    </BODY>
    </HTML>
```

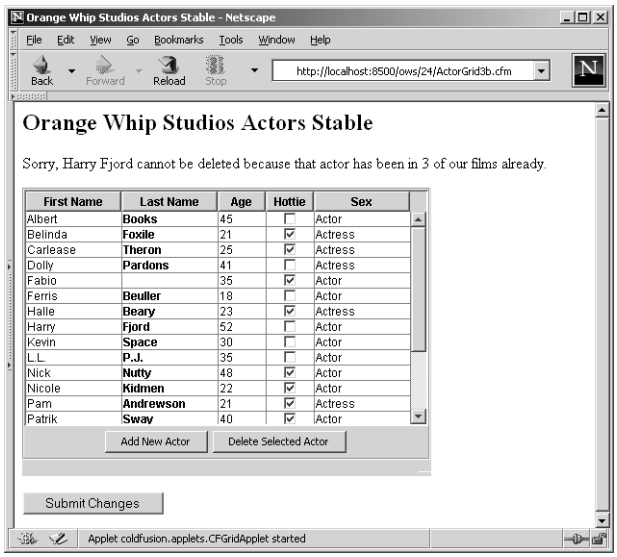
First, the `NumChanges` variable is set to the length of the `FORM.ActorGrid.RowStatus.Action` array. This is the number of edits (inserts, updates, or deletes) the user made to the grid. If the user submitted the form without making any changes, the value of `NumChanges` is `0`.

Next, a `<CFLOOP>` tag is used to loop over each of the elements in the special arrays. For instance, for each iteration of the loop, the value of `RowStatus.Action[i]` is `U`, `D`, or `I` to indicate an update, delete, or insert. This value is stored in the `ThisAction` variable. A `<CFSWITCH>` block is then used to execute a different `<CFCASE>` block for each of the possible values of `ThisAction` (`U`, `D`, or `I`).

Within each of the <CFCASE> tags, the original and updated values for each column of the grid are available for use in queries or other CFML code. For instance, within the <CFCASE> for deletes, the value of FORM.ActorGrid.original.ActorID[i] is used to determine which actor's record was deleted from the grid. A quick query named GetActorFilms is used to determine whether the actor is already in any films. If the actor is not in any films, the record is deleted. If the actor is in at least one film, the deletion is skipped (Figure 24.7).

Figure 24.7

If the user attempts to delete an actor whose ID number is already used in some other table, the deletion is cancelled.



Conditional Color-Coding

<CFGRID> includes the capability to display grid cells in various colors based on the current grid values. Similar conceptually to the Conditional Formatting feature in Microsoft Excel, this gives you a way to visually highlight values that are particularly low, high, or otherwise unusual or noteworthy. The color-coding is applied in real time, so if the grid is updatable, cell colors can change while the user is editing.

To use the conditional color-coding feature, you use a special formula syntax in the TEXTCOLOR or BGCOLOR attribute of the <CFGRIDCOLUMN> tag (refer to Table 24.5). This formula looks similar to CFML and JavaScript and always must appear within parentheses, as shown in the following. The formula syntax is simple and is really much easier to show than to describe. Consider the following:

```
TEXTCOLOR="(CX GT 25 ? black : red)"
```

This would cause each cell in the column to appear in black when the current value of the cell was more than 25 and in red when less than or equal to 25. The CX stands for the value of the current cell. The GT means greater than, just like in CFML. The ? and : characters are used to specify

the colors that should be used if the condition is met or not met, respectively. The color can be specified as a named color or as a hexadecimal RGB color value, like so:

```
TEXTCOLOR="(CX GT 25 ? 000000 : FF0000)"
```

You can use a numbered column number in place of the `CX`. The first column is `C0`, the second is `C1`, and so on. This enables you to set a cell's text color based on the values in other cells in the same row of the grid. So, this expression would set the cell's color based on the current value of the third column in the grid:

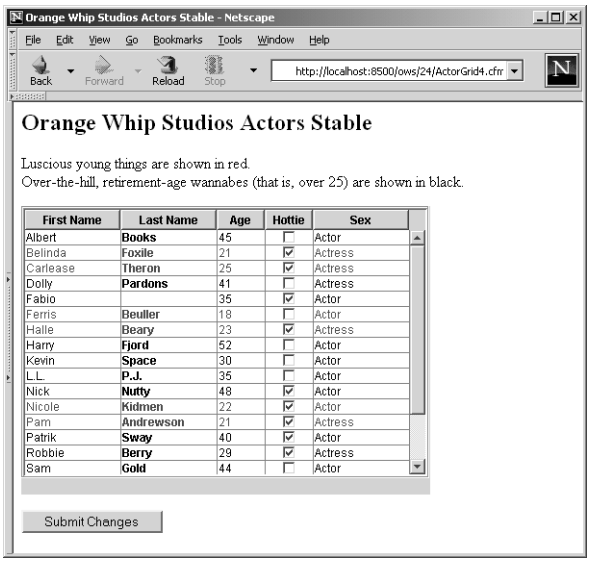
```
TEXTCOLOR="(C2 GT 25 ? black : red)"
```

You can also use `LT` (less than) or `EQ` (equal to) in place of the `GT` in the previous snippet.

Listing 24.9 shows how to use the conditional formatting feature in your code. This example displays actors who are `25` or older in black and younger actors in red. If the user changes the age column for an actor, the cells in that row will be colored accordingly (Figure 24.8).

Figure 24.8

Conditional formatting can be used to color-code cells in real time as the user edits the grid.



TIP

When using conditional formatting, be sure to give your users some type of legend or explanation to help them understand what each color means. This listing provides such an explanation at the top of the page (see Figure 24.8).

NOTE

You might not be able to see the difference between the red and black type in Figure 24.8 as printed in this book, but you will see the difference if you visit this listing with your browser.

Listing 24.9 `ActorGrid4.cfm`—Highlighting Certain Values with Conditional Formatting

```
<!---
  Filename: ActorGrid4.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Demonstrates use of the <CFGRID> tag
--->


<!--- If form is being submitted, commit changes to the database --->
<CFIF CGI.REQUEST_METHOD is "Post">
  <CFGRIDUPDATE
    DATASOURCE="ows"
    TABLENAME="Actors"
    GRID="ActorGrid"
    KEYONLY="No">
</CFIF>

<!--- Get list of Actors from database --->
<CFQUERY NAME="GetActors" DATASOURCE="ows">
  SELECT * FROM Actors
  ORDER BY NameFirst, NameLast
</CFQUERY>


<HTML>
<HEAD><TITLE>Orange Whip Studios Actors Stable</TITLE></HEAD>
<BODY>
<H2>Orange Whip Studios Actors Stable</H2>

Luscious young things are shown in red.<BR>
Over-the-hill, retirement-age wannabes (that is, over 25) are shown in black.<BR>

<!--- Data entry form --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="Post">

  <!--- Grid control for expenses --->
  <CFGRID
    NAME="ActorGrid"
    QUERY="GetActors"
    WIDTH="425"
    HEIGHT="300"
    ROWHEADERS="No"
    COLHEADERBOLD="Yes"
    COLHEADERALIGN="CENTER"
    SELECTMODE="EDIT">
    <!--- Hidden column for ActorID - does not actually get displayed --->
    <CFGRIDCOLUMN
      NAME="ActorID"
      DISPLAY="No">
    <!--- Column for actor's first name --->
    <CFGRIDCOLUMN
      NAME="NameFirst"
      HEADER="First Name"
      WIDTH="100"
      TEXTCOLOR="(C2 GT 25 ? black : red )">
```

Listing 24.9 (CONTINUED)

```
        <!--- Column for actor's last name --->
        <CFGRIDCOLUMN
          NAME="NameLast"
          HEADER="Last Name"
          WIDTH="100"
          BOLD="Yes"
          TEXTCOLOR="(C2 GT 25 ? black : red )">
        <!--- Column for actor's age--->
        <CFGRIDCOLUMN
          NAME="Age"
          HEADER="Age"
          WIDTH="50"
          NUMBERFORMAT="999."
          TEXTCOLOR="(CX GT 25 ? black : red )">
        <!--- Column for babe factor --->
        <CFGRIDCOLUMN
          NAME="IsTotalBabe"
          HEADER="Hottie"
          WIDTH="50"
          TYPE="boolean">
        <!--- Column for gender --->
        <CFGRIDCOLUMN
          NAME="Gender"
          HEADER="Sex"
          WIDTH="100"
          VALUES="M,F"
          VALUESDISPLAY="Actor,Actress"
          TEXTCOLOR="(C2 GT 25 ? black : red )">
      </CFGRID>

      <P><INPUT TYPE="Submit" VALUE="Submit Changes">
    </CFFORM>

    </BODY>
    </HTML>
```

NOTE

This example uses conditional formatting for the `TEXTCOLOR` attribute only, but you can also use it with the `BGCOLOR` attribute (using the same formula syntax).

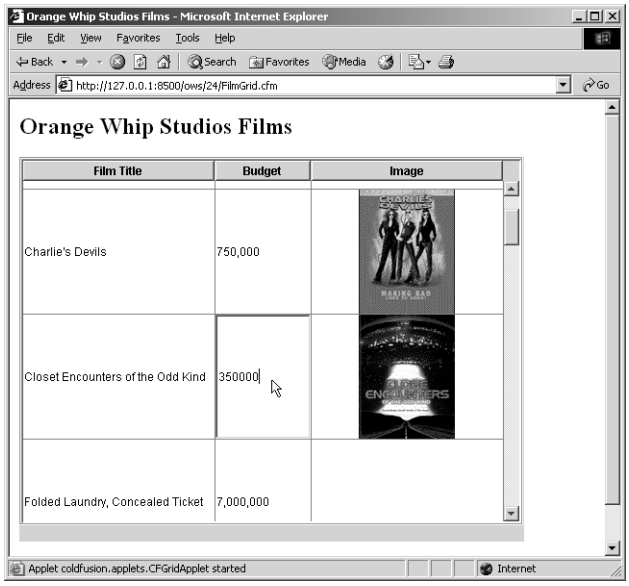
Displaying Images in Grids

The `<CFGRID>` tag is capable of displaying images in individual cells. To display images in a grid column, set the `TYPE` attribute of the `<CFGRIDCOLUMN>` to `Image`; then make sure that the values in the corresponding column of your query contain URLs for the appropriate images to display in each cell. If the grid is editable, the user will be able to double-click the image to edit the path and submit the changes to the server.

Listing 24.10 demonstrates how to use `TYPE="Image"` to display images in a grid. Each film record is shown with the appropriate image, if one is currently available for the film. If no image exists for the film, the `Image` column appears as empty (Figure 24.9).

Figure 24.9

GIF and JPEG images can be displayed in grids, along with other types of information.



Listing 24.10 `FilmGrid.cfm`—Displaying Images in an Editable Grid

```
<!---
  Filename: FilmGrid.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Demonrates use of <CFGRID> to display images
--->


<!--- If form is being submitted, commit changes to the database --->
<CFIF CGI.REQUEST_METHOD is "Post">
  <CFGRIDUPDATE
    DATASOURCE="ows"
    TABLENAME="Actors"
    GRID="ActorGrid"
    KEYONLY="No">
</CFIF>

<!--- Get list of films from database --->
<!--- Include a column of blank values called ImagePath --->
<CFQUERY NAME="GetFilms" DATASOURCE="ows">
  SELECT FilmID, MovieTitle, AmountBudgeted, '' AS ImagePath
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- For each film, check to see if we have a image for it --->
<CFLOOP QUERY="GetFilms">
  <!--- This is the relative URL path for the image --->
  <CFSET TestImagePath = "../images/f#GetFilms.FilmID#.gif">
  <!--- If the image file actually exists on this server's drive --->
  <CFIF FileExists(ExpandPath(TestImagePath))>
    <!--- Set the ImagePath column of the current row --->
```

Listing 24.10 (CONTINUED)

```
        <CFSET GetFilms.ImagePath[CurrentRow] = TestImagePath>
      </CFIF>
    </CFLOOP>

    <HTML>
    <HEAD><TITLE>Orange Whip Studios Films</TITLE></HEAD>
    <BODY>
    <H2>Orange Whip Studios Films</H2>

    <!--- Data entry form --->
    <CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="Post">

      <!--- Grid control for expenses --->
      <CFGRID
        NAME="ActorGrid"
        QUERY="GetFilms"
        WIDTH="525"
        HEIGHT="400"
        ROWHEADERS="No"
        ROWHEIGHT="130"
        COLHEADERBOLD="Yes"
        COLHEADERALIGN="Center"
        SELECTMODE="EDIT">
        <!--- Hidden column for FilmID - does not actually get displayed --->
        <CFGRIDCOLUMN
          NAME="FilmID"
          DISPLAY="No">
        <!--- Column for film's title --->
        <CFGRIDCOLUMN
          NAME="MovieTitle"
          HEADER="Film Title"
          WIDTH="200">
        <!--- Column for film's budget --->
        <CFGRIDCOLUMN
          NAME="AmountBudgeted"
          HEADER="Budget"
          WIDTH="100"
          NUMBERFORMAT="999,999,999,999.">
        <!--- Column for film's publicity image, if any --->
        <CFGRIDCOLUMN
          TYPE="Image"
          NAME="ImagePath"
          HEADER="Image"
          WIDTH="200">

      </CFGRID>

      <P><INPUT TYPE="Submit" VALUE="Submit Changes">
    </CFFORM>

    </BODY>
    </HTML>
```

First, a query called `GetFilms` is executed. Simple `AS` syntax is used to add another column called `ImagePath` to the query as it is being returned to ColdFusion. This column will be filled with the relative URL to each film's image, if an image is currently available for the film.

Next, a <CFLOOP> block is used to iterate over each row of the GetFilms query. Within the loop, the TestImagePath variable is set to what the relative path for the film's image should be, which is based on the film's ID number. The ExpandPath() and FileExists() functions are then used to determine whether an image file actually exists for the film; if so, the URL path for the image is placed in the ImagePath column of the query. When the loop has finished its work, each film that has an image will have the relative path to the image (starting with ../images) as the ImagePath column; other films will still have an empty string in that column.

Finally, the ImagePath column name is passed as the NAME attribute of a <CFGRIDCOLUMN> tag of TYPE="Image", which causes the images to be displayed in the grid (as shown in Figure 24.9).

About the <CFGRIDROW> Tag

Up to this point, you've been dealing with grids that get their data from a query. You also can add rows to a grid without using a query by omitting the QUERY attribute from the <CFGRID> tag and then adding a <CFGRIDROW> tag between the <CFGRID> tag pair.

I would recommend against using the <CFGRIDROW> tag. The main problem is that it requires you to supply all data for the row as a comma-separated list, in which the commas indicate where the data should be split into separate columns. However, in contrast to most uses of lists within ColdFusion, you have no opportunity to supply an alternative delimiter character, which leads to problems if any of the row's values are empty strings or contain commas. It's too bad that <CFGRIDROW> doesn't take a COLUMN attribute; that would solve the problem and turn it into a useful tag.

The good news is that there is really no need for <CFGRIDROW> in the first place. If you need to supply nondatabase data to a query, simply create a query object yourself, using the QueryNew(), QueryAddRow(), and QuerySetCell() functions. You can read more about these functions in Appendix C; there is also an example in Chapter 27, "Online Commerce."

If you want find out more about the <CFGRIDROW> tag, consult the CFML Reference section of the ColdFusion documentation.

Using <CFSLIDER>

The <CFSLIDER> tag places a sliding bar control on a page that enables you to select a numeric value by moving a knob from side to side (or up and down). This type of control is ideal for situations in which you need to collect some kind of rating or ranking from your users. The slider control can be defined with a number of settings for its range, alignment, default value, and other features.

Table 24.8 is an abbreviated list of the attributes supported by the <CFSLIDER> tag. We've shortened the list here to make it easier for you to concentrate on the most important attributes. The complete list is provided in Appendix B.

NOTE

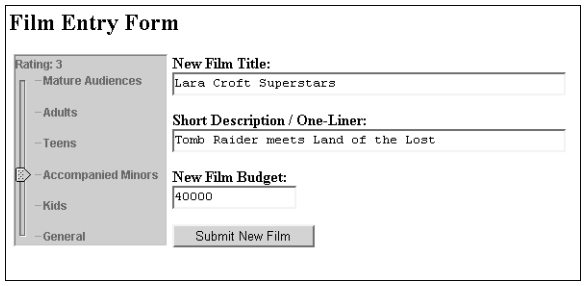
The <CFSLIDER> tag also supports most of the look-and-feel attributes listed in Table 24.3, earlier in this chapter.

Table 24.8 Abbreviated List of <CFSLIDER> Attributes

ATTRIBUTES	DESCRIPTION
NAME	Required. A name for the slider, which must be a unique name within the form. When the form is submitted, the slider's value will be available as the FORM variable indicated by this name.
VALUE	Optional. The value that should be preselected in the slider when the page first appears. If omitted, defaults to the minimum RANGE value (see the following).
RANGE	Optional. Determines the minimum and maximum values selectable with the slider. Separate values by a comma. For example RANGE="0,500". Default is 0,100. Valid only for numeric data.
SCALE	Optional. A number that indicates which values the slider should be capable of being set to within the RANGE. So, if the RANGE is 0,500 and the SCALE is 50, the user would be able to use the slider to choose only 0, 50, 100, 150, and so on. If omitted, the user can select any value within the RANGE.
LABEL	Optional. A label that appears on the slider control so the user can easily see the numeric value he has selected by sliding the control's knob. Use the word %value% (surrounded by percent signs, as shown) to represent the current value of the slider, like so: LABEL="Volume: %value%". Remember to set the REFRESHLABEL attribute to Yes whenever you provide a LABEL attribute.
REFRESHLABEL	Optional. Yes or No. If Yes, the label is not refreshed when the slider is moved. Default is Yes.
VERTICAL	Optional. Yes or No. If No, the slider knob can be moved from side to side. If Yes, the knob moves up and down. Default is No.
TICKMARKMAJOR	Optional. Yes or No. If Yes, large tick marks appear along the slider control to help the user see where particular values will fall. Default is No. There is also a TICKMARKMAJOR attribute (see Appendix B).
TICKMARKLABELS	Optional. If No (the default), no labels are shown along the slider control to mark where particular values fall. If Yes or Numeric, labels are automatically displayed based on the RANGE and SCALE attributes. If any other value is supplied, it is assumed to be a comma-separated list of tick labels to display (see Listing 24.11 for an example).

To use the slider control, just add the <CFSLIDER> tag to your <CFFORM> code. Most of the time, you will want to specify values for (at least) the NAME, WIDTH, HEIGHT, and RANGE attributes (see Table 24.8). Listing 24.11 shows how to use the slider control in your ColdFusion templates (Figure 24.10).

Figure 24.10
The <CFSLIDER> tag emphasizes the relationship between the values in a numeric range.



Listing 24.11 `FilmEntry.cfm`—Adding a Slider Control to a Data-Entry Form

```
<!---
  Filename: FilmEntry.cfm
  Author:   Nate Weiss (NMW)
  Purpose:  Data entry interface with slider selection for rating
--->


<!--- Insert film into database when form is submitted --->
<CFIF IsDefined("FORM.RatingID")>
  <CFQUERY DATASOURCE="ows">
    INSERT INTO Films(MovieTitle, RatingID, AmountBudgeted, PitchText)
    VALUES ('#FORM.MovieTitle#', #FORM.RatingID#, #FORM.AmountBudgeted#, ' ')
  </CFQUERY>
</CFIF>

<!--- Get list of ratings from database --->
<CFQUERY NAME="GetRatings" DATASOURCE="ows">
  SELECT RatingID, Rating
  FROM FilmsRatings
  ORDER BY RatingID
</CFQUERY>

<!--- Use query of queries feature to get highest and lowest rating ID --->
<CFQUERY NAME="GetMinMax" DBTYPE="query">
  SELECT MIN(RatingID) AS MinRating, MAX(RatingID) As MaxRating
  FROM GetRatings
</CFQUERY>

<HTML>
<HEAD><TITLE>Film Entry Form</TITLE></HEAD>
<BODY>
<H2>Film Entry Form</H2>

<!--- Data entry form --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="POST">
  <!--- Slider control for selecting the film's rating --->
  <CFSLIDER
    NAME="RatingID"
    VALUE="2"
    LABEL="Rating: %value%"
    RANGE="#GetMinMax.MinRating#,#GetMinMax.MaxRating#"
    TICKMARKMAJOR="Yes"
    TICKMARKLABELS="#ValueList(GetRatings.Rating)#"
    VERTICAL="Yes"
    WIDTH="160"
    HEIGHT="200"
    LOOKANDFEEL="Metal"
    ALIGN="LEFT"
    HSPACE="5">

  <!--- Text entry field for expense description --->
  <P><B>New Film Title:</B><BR>
  <CFINPUT NAME="MovieTitle" SIZE="50" MAXLENGTH="50"
    REQUIRED="Yes" MESSAGE="Please don't leave the film title blank."><BR>
```

Listing 24.11 (continued)

```
    <!--- Text entry field for expense description --->
    <P><B>Short Description / One-Liner:</B><BR>
    <CFINPUT NAME="PitchText" SIZE="50" MAXLENGTH="50"
      REQUIRED="Yes" MESSAGE="Please don't leave the one-liner blank."><BR>

    <!--- Text entry field for expense description --->
    <P><B>New Film Budget:</B><BR>
    <CFINPUT NAME="AmountBudgeted" SIZE="15"
      REQUIRED="Yes" MESSAGE="Please don't leave the film title blank."
      VALIDATE="float"><BR>



    <!--- Submit button for form --->
    <P><INPUT TYPE="Submit" VALUE="Submit New Film">
  </CFFORM>

  </BODY>
  </HTML>
```

About <CFTEXTINPUT> and <CFAPPLET>

This section describes two <CFFORM> tags that were not covered in detail in this chapter: <CFTEXTINPUT> and <CFAPPLET>. Because these tags aren't generally used in the majority of ColdFusion applications, we aren't providing specific examples for them here. You can find out more about them in the ColdFusion documentation.

About <CFTEXTINPUT>

In this chapter, you have learned about the sophisticated Java-based controls provided by ColdFusion for use in your forms (the tree control, grid control, and slider control). ColdFusion also provides a Java-based alternative to HTML text form fields, called <CFTEXTINPUT>. It provides all the validation options provided by the <CFINPUT> tag you learned about in Chapter 12, plus a number of additional attributes to control font color, size, and so on.

In general, we don't recommend that you use this tag, so we aren't providing a specific code example for it in this chapter. You can get all the same validation functionality from <CFINPUT> without the added overhead of Java, and recent versions of the major browsers allow you to specify font size and color for text form fields via CSS formatting.

To learn more about <CFTEXTINPUT>, consult Appendix B or the CFML Language Reference section of the ColdFusion documentation.

About <CFAPPLET>

One of the least used or understood ColdFusion form features is the <CFAPPLET> tag. This tag enables you to extend the power of <CFFORM> by adding new Java elements that you create (or acquire or buy from other companies or developers). You can have the results of the Java applet submitted with the

form, as if it were an ordinary form element. In addition, much of the code you normally would write for a Java applet is handled by ColdFusion using default values you set during registration.

Because the writing of Java applets is a topic well beyond the scope of this book, and because free, prebuilt Java controls that can effectively be used as form controls are in short supply, we aren't providing any specific <CFAPPLET> examples in this chapter. In general, you will be provided with prebuilt syntax for using a particular applet in your code, so the services provided by the <CFAPPLET> tag are not generally of much real-world benefit.

To use <CFAPPLET>, you first add an entry in the Java Applets page of the ColdFusion Administrator, and then use the <CFAPPLET> tag in your form templates. For more information, consult Appendix B or the ColdFusion documentation.