CHAPTER 26

Integrating with Macromedia Flash

In this chapter

USING THE FLASH COMPONENT KIT

Macromedia provides a set of prebuilt user interface components that make including certain types of visual widgets in your ColdFusion pages remarkably easy. These stylish, highly interactive components are implemented as Macromedia Flash movies. The Macromedia Flash 5 Player—which must be present on the user's machine—takes care of presenting the components to the user.

The components are available as a part of the Macromedia Flash Component Kit for ColdFusion, which Macromedia makes available to developers free of charge. The latest available version of the kit has been included on this book's CD-ROM for your convenience. Updates will be made available from the ColdFusion Developers Exchange site at `http://devex.allaire.com/developer/gallery` (search for Flash Component Kit).

INTRODUCING THE FLASH COMPONENT KIT

The Macromedia Flash Component Kit for ColdFusion serves two purposes:

- **It provides a set of prebuilt, visual components, which you can use in your pages**—Each component is encapsulated within a custom tag, which makes them very easy to use. Sample templates that use each of the components are provided in this chapter.
- **It also provides tips, technical resources, and best practices information about creating additional components of your own**—This topic is beyond the scope of this chapter, but you are encouraged to look through the PDF documents provided with the Component Kit to learn more about creating your own user-interface widgets with Macromedia Flash.

Currently, four prebuilt components are included in the Flash Component Kit:

- **The Calendar component**—Used for letting your users pick dates visually
- **The Calculator component**—Embeds a calculator on your Web pages
- **The Navigation Bar component**—Used for placing an interactive toolbar at the top of your application pages
- **The IE Cascading Menu component**—A more advanced type of navigation bar

All the components provide slick-looking, interactive widgets, which can be used to give a more exciting, dynamic feel to your application. Because they rely on the Macromedia Flash Player to provide the interactivity, they work exactly the same way in nearly all browsers that support the Flash Player (except for the Cascading Menu component, which works with Internet Explorer on Windows only).

Note

In general, you could build similar components using Dynamic HTML, but developing Dynamic HTML that works properly in all the major browsers is usually quite difficult. That said, a great place to look for cross-browser Dynamic HTML solutions is `http://www.webreference.com`.

INSTALLING THE COMPONENT KIT

To install the Component Kit, follow these steps:

1. The Component Kit is provided in the form of a Zip archive named `componentkit.zip` (for use on Windows servers) or a TAR archive named `componentkit.tar` (for Linux/Unix users). Extract the appropriate archive to a temporary directory, making sure that the archive's directory structure is maintained during the extraction process.
2. Copy the entire `componentkittags` folder from the extracted archive to the `CustomTags` folder, which is located within the `CFUSION` folder. There should now be a `CFUSION\CustomTags\componentkittags` folder, with a number of ColdFusion templates in it.
3. Create a new folder called `componentkit` in your Web server's root directory.
4. Copy the `support` and `samples` folders from the extracted archive to the `componentkit` folder you just created.

Note

You are free to use a different folder name instead of `componentkit`. Actually, you can use any folder location you want, as long as it is accessible via your Web server. If you use a different folder location, just be sure to provide the relative path to that location in the `uiToolkitCFPath` and `uiToolkitSupportPath` variables later (see Listing 26.1).

To test the installation, visit the following URL with your Web browser (if you installed the Component Kit onto a ColdFusion server other than your local machine, use that server's name or IP address instead of `localhost`):

```
http://localhost/componentkit/samples/index.htm
```

An introductory page should appear, with links to live examples of the four prebuilt components included with the kit. At this point, you should be able to interact with the examples and see components in action. If not, make sure the Flash Player 5 (or later) is installed on your browser machine; then try going through the installation process on the server machine again.

USING THE CALENDAR COMPONENT

The Calendar component is a handy date-picking widget you can use to collect dates from your users. Rather than typing a date into a plain text box, your users can interact with a visual calendar. This makes it a lot easier for them to understand which day of the week they are committing to, and to generally get a better sense of how far away from the present the dates might actually be.

From your perspective as a developer, the Calendar component behaves similarly to the form fields you have seen throughout this book. Used in a form page, it collects date information from the user and makes the date value available to the receiving template when the form is submitted. The Calendar component will usually be used within a <CFFORM> tag.

To include the Calendar in one of your pages, use the <CF_uicalendar> tag. Table 26.1 lists the attributes this tag supports.

TABLE 26.1 <CF_uicalendar> TAG ATTRIBUTES

Attribute	Description
FORMFIELD	A name for the component's value, which will be available to the form's ACTION page when the form is submitted. This attribute is equivalent to the NAME attribute of an ordinary form control, such as <INPUT>, <TEXTAREA>, or <CFINPUT>.
POPUP	Yes or No. If Yes, which is the default, the user has an ordinary text field for entering the date. The text field has a small calendar icon next to it, which the user can use to invoke the calendar. If No, the calendar appears right away, without any separate text entry field next to it.
MONTH	The month that should be shown by default when the calendar first appears. If this attribute is not provided, the current month is used.
DAY	The day that should appear selected when the calendar first appears. If it's not provided, the current day is used.
YEAR	The year that should be shown when the calendar first appears. If it's not provided, the current year is used.
DATEFORMAT	The format used to display (and submit) the date the user selects with the calendar control. Unfortunately, you cannot specify a date mask here, as with the DateFormat() function. Instead, the only choices are date and eurodate, which correspond to the date and eurodate values for the VALIDATE attribute of the <CFINPUT> tag.
WIDTH	The width of the calculator, in pixels.
HEIGHT	The height of the calculator, in pixels.
stFORMATTING	A ColdFusion structure containing the generic formatting properties supported by the Flash Component Kit. See Listing 26.3, later in this chapter, for details.
SELBDRCOLOR	A color for the border that indicates the currently selected date. Provide the value in RGB hexadecimal format, preceded by two # signs, such as ##FFFFFF for white or ##0000FF for blue. You can't use named colors, such as red or blue.
ZINDEX	A zindex value for the calculator component, which can be used to control whether the calculator appears in front of or behind other dynamic elements on the page. If it's not provided, a default value of 100 is used; use a higher number if the calendar appears behind other absolutely positioned elements on the page. Currently relevant only when the Calendar is viewed with Internet Explorer. For more information about zindex and absolutely positioned elements, refer to a DHTML reference book or the DHTML References section of http://msdn.microsoft.com/.

Attribute	Description
REQUIRED	Undocumented. Yes or No. If Yes, the user must provide a date before the form can be submitted. Corresponds to the REQUIRED attribute of the <CFINPUT> tag. This attribute is relevant only when POPUP="Yes". It has no effect if POPUP="No".
MESSAGE	Undocumented. The message to be shown if REQUIRED="Yes" and the user leaves the value blank. Corresponds to the MESSAGE attribute of the <CFINPUT> tag. If it's omitted, a default message (Invalid Date Format) is used. This attribute is relevant only when POPUP and REQUIRED are both Yes.

Note

The attributes marked as Undocumented are implemented in the custom tags provided in the Component Kit but are not documented for some reason. They work in the version of the Component Kit that was available when this book was written. There is no explicit guarantee that they will continue to work in future versions of the kit.

GETTING READY TO USE THE CALENDAR

As you are about to see, including the Calendar component in your application pages is very easy.

Before you can do so, you must set a few special variables in your Application.cfm file. All the components in the Flash Component Kit rely on these variables to locate the various images, Flash movie (.swf) files, and other elements used to display the components to your users.

The required variables are

- REQUEST.uiToolkitCFPath—This should always be set to /componentkit/support/, as long as you used the suggested folder names while installing the Component Kit. If you used a different folder location when installing, provide the relative path to the Component Kit's support folder. The <CF_uicalendar> and other custom tags provided by the Component Kit tags will use this value internally in <CFINCLUDE> tags, which means that any mappings established in the ColdFusion Administrator will be respected.
- REQUEST.uiToolkitSupportPath—This should also always be set to /componentkit/support/ as long as you used the suggested folder names while installing the Component Kit. If you used a different folder location when installing, provide the relative URL path to the Component Kit's support folder. The <CF_uicalendar> and other custom tags provided by the Component Kit tags will use this value internally in the SRC for images and script files, and in the MOVIE path for the Flash Player. Therefore, any folder or virtual Web server mappings established at the Web server level will be respected.

Note

To learn how the REQUEST scope might be helpful to use in your own custom tags, see Chapter 22, "Building Reusable Components."

Note

In most situations, the values for REQUEST.uiToolkitCFPath and REQUEST.uiToolkitSupportPath are the same.

Listing 26.1 is a typical Application.cfm template, with the two required variables added. The variables have each been set to /componentkit/support/ under the assumption that you installed the Component Kit using the suggested folder locations.

Save this file as Application.cfm, not Application1.cfm.

LISTING 26.1 Application1.cfm—ADDING THE COMPONENT KIT'S REQUIRED VARIABLES TO THE REQUEST SCOPE

```
<!---
  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">

<!--- These variables required by the Flash Component Kit --->
<CFSET REQUEST.uiToolkitCFPath = "/componentkit/support/">
<CFSET REQUEST.uiToolkitSupportPath = "/componentkit/support/">
```

Note

The Component Kit does not specifically require that you define these variables in Application.cfm, but it does require that the variables be set in the REQUEST scope before you use any of the components in a ColdFusion template. The easiest way to do this is to set them in Application.cfm, but you could just set them at the top of each page that uses a Component Kit component instead.

Note

Like all file paths, the values of the uiToolkitCFPath and uiToolkitSupportPath variables are case sensitive on Unix/Linux servers. They are not case sensitive on Windows servers.

USING THE POP-UP CALENDAR FOR DATA ENTRY

Now that the required variables have been added to the REQUEST scope, you are free to use the Calendar component in your application pages. Simply create a form page using <CFFORM>, and then place the <CF_uicalendar> tag between the <CFFORM> tags.

Listing 26.2 shows how to use the Calendar component in a basic data-entry form. This example creates a simple form for inserting new records in the Films table. The user uses conventional text entry fields to provide the new film's MovieTitle, PitchText, and AmountBudgeted values. The Calendar component assists the user in providing the DateInTheaters value, as shown in Figure 26.1.

LISTING 26.2 FilmEntry1.cfm—USING THE CALENDAR COMPONENT

```
<HTML>
<HEAD><TITLE>New Film</TITLE></HEAD>
<BODY>
<H2>Adding New Film</H2>

<!--- If User is Submitting the Form --->
<CFIF IsDefined("FORM.MovieTitle")>
  <!--- Insert new record into Films table in database --->
  <CFINSERT
    DATASOURCE="#REQUEST.DataSource#"
    TABLENAME="Films">

  <!--- Display success message --->
  <CFOUTPUT>
    <B>#FORM.MovieTitle#</B> was added.<BR>
  </CFOUTPUT>
</CFIF>

<!--- Self-submitting data-entry form --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="POST">
  <TABLE>
    <!--- Text entry field for: MovieTitle --->
    <TR>
      <TH>Film Title:</TH>
      <TD>
        <CFINPUT
          NAME="MovieTitle"
          SIZE="30"
          MAXLENGTH="50"
          REQUIRED="Yes"
          MESSAGE="You may not leave the title blank.">
      </TD>
    </TR>

    <!--- Text entry field for: MovieTitle --->
    <TR>
      <TH>One-Liner:</TH>
      <TD>
        <CFINPUT
          NAME="PitchText"
          SIZE="60"
```

LISTING 26.2 CONTINUED

```
          MAXLENGTH="200"
          REQUIRED="Yes"
          MESSAGE="You may not leave the One-Liner blank.">
      </TD>
    </TR>

    <!--- Text entry field for: AmountBudgeted --->
    <TR>
      <TH>Film Budget:</TH>
      <TD>
        <CFINPUT
          NAME="AmountBudgeted"
          SIZE="10"
          MAXLENGTH="200"
          VALIDATE="float"
          REQUIRED="Yes"
          MESSAGE="You may not leave the Film Budget blank.">
      </TD>
    </TR>

    <!--- Text entry field for: DateInTheaters --->
    <!--- Augmented with Flash Calendar component from FCK --->
    <TR>
      <TH>Release Date:</TH>
      <TD>
        <CF_uicalendar
          FORMFIELD="DateInTheaters"
          WIDTH="200"
          HEIGHT="200"
          POPUP="Yes"
          SELBDRCOLOR="##FFFF00"
          REQUIRED="Yes"
          MESSAGE="You must provide a release date.">
      </TD>
    </TR>

  </TABLE>

  <!--- Submit Button to create new film record --->
  <P><INPUT TYPE="Submit" VALUE="Insert New Film Record Now">
</CFFORM>

</BODY>
</HTML>
```

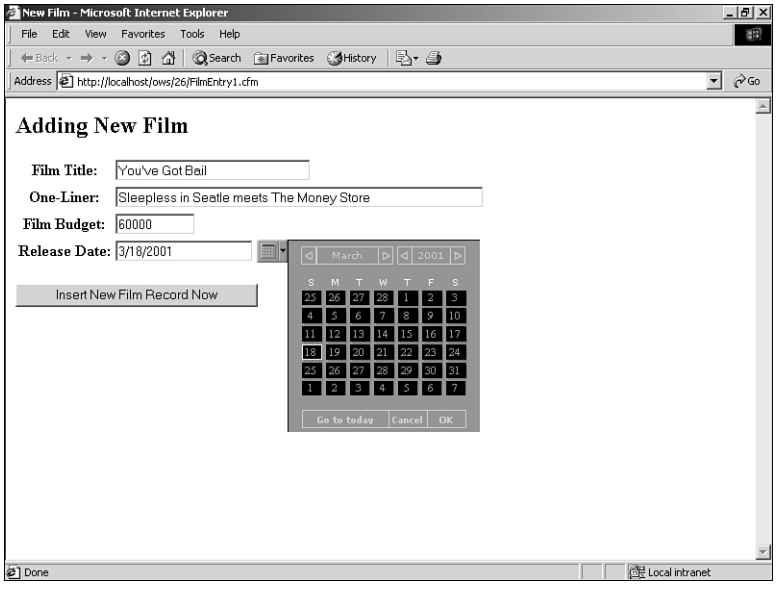

Figure 26.1
The Calendar Component provides a convenient way for your users to enter dates.

Note

To use the Calendar with `POPUP="Yes"`, you must place the `<CF_uicalendar>` tag between opening and closing `<CFFORM>` tags. For more information about `<CFFORM>`, see Chapter 13, "Form Data Validation," and Chapter 25, "Enhancing Forms with Client-Side Java."

CONTROLLING THE LOOK AND FEEL

The Flash Component Kit gives you a lot of flexibility when it comes to customizing the look of the individual controls. To adjust properties such as the color and fonts used in a control, create a ColdFusion structure that contains the properties you want to adjust. The structure can contain any or all of the properties shown in Table 26.2. Then provide the structure to the optional `stFORMATTING` attribute of the `<CF_uicalendar>` or other Component Kit controls.

TABLE 26.2 FORMATTING ATTRIBUTES

Structure Attribute	Description
`bgcolor`	Component background color, as an RGB hexadecimal value.
`bgtrans`	Whether to set background transparency as a Boolean value. Currently, it works only with Internet Explorer and not on all platforms.
`bdrstate`	Whether to display a border as a Boolean value.
`bdrcolor`	The border color, as an RGB hexadecimal value.
`btncolor`	Background color for buttons, as an RGB hexadecimal value.
`btntxtcolor`	Text color for buttons, as an RGB hexadecimal value.

TABLE 26.2 CONTINUED

Structure Attribute	Description
`btnbdrcolor`	Border color for buttons, as an RGB hexadecimal value.
`btnhicolor`	Background color to be used when a button is clicked, as an RGB hexadecimal value.
`btntxthicolor`	Text color to be used when a button is clicked, as an RGB hexadecimal value.
`btnbdrhicolor`	Border color to be used when a button is clicked, as an RGB hexadecimal value.
`txtcolor`	Text color, as an RGB hexadecimal value.
`txtfont`	Text font name, such as `Arial` or `Helvetica`. You can also use the generic `_sans`, `_serif`, or `_typewriter` font names, which are defined by the Flash Player and should look the same on all machines, regardless of which fonts are actually installed on the user's machine.
`rdonly`	Whether component should be considered read-only, as a Boolean value.
`submitcontrols`	Whether the component should include OK and Cancel buttons.

You can create the formatting structure right before using each component, in your `Application.cfm` file, or in a separate file that you include via `<CFINCLUDE>` before using a component. (For more information about `<CFINCLUDE>`, see Chapter 10, "CFML Basics.")

Listing 26.3 is a revised version of `Application.cfm`. It creates a formatting structure called `REQUEST.uiToolkitFormatting` to create an attractive look for the Calendar component, using shades of blue. Save this listing as `Application.cfm`, not `Application2.cfm`.

LISTING 26.3 `Application2.cfm`—DEFINING A LOOK AND FEEL FOR COMPONENT KIT WIDGETS

```
<!---
  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">

<!--- These variables required by the Flash Component Kit --->
<CFSET REQUEST.uiToolkitCFPath = "/componentkit/support/">
<CFSET REQUEST.uiToolkitSupportPath = "/componentkit/support/">

<!--- Formatting Structure for Flash Component Kit components --->
<CFSET REQUEST.uiToolkitFormatting = StructNew()>
<CFSET REQUEST.uiToolkitFormatting.txtfont = "_sans">
<CFSET REQUEST.uiToolkitFormatting.bgcolor = "##0000CD">
<CFSET REQUEST.uiToolkitFormatting.btncolor = "##FFFFFF">
```

```
<CFSET REQUEST.uiToolkitFormatting.btntxtcolor = "##000080">
<CFSET REQUEST.uiToolkitFormatting.btnbdrcolor = "##0000FF">
<CFSET REQUEST.uiToolkitFormatting.bdrcolor = "##0000FF">
```

Now that the formatting structure has been set up, you just need to go back to Listing 26.2 and add `stFORMATTING="#REQUEST.uiToolkitFormatting#"` to the `<CF_uicalendar>` tag. When you reload Listing 26.2 in your browser, you will see that your formatting instructions have been applied.

USING THE INLINE CALENDAR

You can also use the Calendar in an inline mode, using `POPUP="No"` in the `<CF_uicalendar>` tag. The Calendar will appear on the page right away, with the `WIDTH` and `HEIGHT` that you indicate.

Unlike ordinary images, the Calendar will appear on top of (obscuring) the other elements on the page (like text). Therefore, you need to be sure to leave space for the Calendar to appear. An easy way to do this is to place the Calendar in its own table cell that has the same `WIDTH` and `HEIGHT` as the component itself. This will reserve the correct amount of space in the page layout for the Calculator to appear in.

Unfortunately, the undocumented `REQUIRED` and `MESSAGE` attributes of the Calendar have no effect when `POPUP="No"` (which might be part of why they are undocumented at this time). You must add a bit of JavaScript to require that the user pick a date (by double-clicking the date) before submitting the form. Using JavaScript for ad-hoc form validation is beyond the scope of this chapter, but it is easy to do. With the help of a JavaScript reference, you will be able to easily adapt the JavaScript code shown in Listing 26.4.

Listing 26.4 demonstrates how to use the inline version of the Calendar. The results are shown in Figure 26.2.

LISTING 26.4 `FilmEntry2.cfm`—USING THE CALENDAR COMPONENT IN INLINE MODE

```
<HTML>
<HEAD>
  <TITLE>New Film</TITLE>
  <!--- Validation script for when the form is submitted --->
  <SCRIPT LANGUAGE="JavaScript">
    function checkForm() {
      var result;
      with (document.forms[0]) {
        // Make DateInTheaters be a required field
        if (DateInTheaters.value == '') {
          alert('You must provide a release date.');
          result = false;
        };
      };
      return result;
    };
  </SCRIPT>
</HEAD>
<BODY>
<H2>Adding New Film</H2>
```

LISTING 26.4 CONTINUED

```
<!--- If User is Submitting the Form --->
<CFIF IsDefined("FORM.MovieTitle")>
  <!--- Insert new record into Films table in database --->
  <CFINSERT
    DATASOURCE="#REQUEST.DataSource#"
    TABLENAME="Films">

  <!--- Display success message --->
  <CFOUTPUT>
    <B>#FORM.MovieTitle#</B> was added.<BR>
  </CFOUTPUT>
</CFIF>

<!--- Self-submitting data-entry form --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="POST" ONSUBMIT="return checkForm()">
  <TABLE>
    <!--- Text entry field for: MovieTitle --->
    <TR>
      <TH>Film Title:</TH>
      <TD>
        <CFINPUT
          NAME="MovieTitle"
          SIZE="30"
          MAXLENGTH="50"
          REQUIRED="Yes"
          MESSAGE="You may not leave the title blank.">
      </TD>
    </TR>

    <!--- Text entry field for: MovieTitle --->
    <TR>
      <TH>One-Liner:</TH>
      <TD>
        <CFINPUT
          NAME="PitchText"
          SIZE="60"
          MAXLENGTH="200"
          REQUIRED="Yes"
          VALUE=""
          MESSAGE="You may not leave the One-Liner blank.">
      </TD>
    </TR>

    <!--- Text entry field for: DateInTheaters --->
    <!--- Augmented with Flash Calculator component from FCK --->
    <TR>
      <TH>Film Budget:</TH>
      <TD>
        <CFINPUT
          NAME="AmountBudgeted"
          SIZE="10"
          MAXLENGTH="200"
          VALIDATE="float"
          REQUIRED="Yes"
          MESSAGE="You may not leave the Film Budget blank.">
      </TD>
```

```
    </TR>

    <!--- Text entry field for: DateInTheaters --->
    <!--- Augmented with Flash Calendar component from FCK --->
    <TR>
      <TH VALIGN="top">Release Date:</TH>
      <TD VALIGN="top" WIDTH="200" HEIGHT="200">
        <CF_uiCALENDAR
          FORMFIELD="DateInTheaters"
          WIDTH="200"
          HEIGHT="200"
          POPUP="No"
          SELBDRCOLOR="##FFFFFF"
          stFORMATTING="#REQUEST.uiToolkitFormatting#"
          REQUIRED="Yes"
          MESSAGE="You must provide a release date.">
      </TD>
    </TR>


  </TABLE>

  <!--- Submit Button to create new film record --->
  <P><INPUT TYPE="Submit" VALUE="Insert New Film Record Now">
</CFFORM>

</BODY>
</HTML>
```

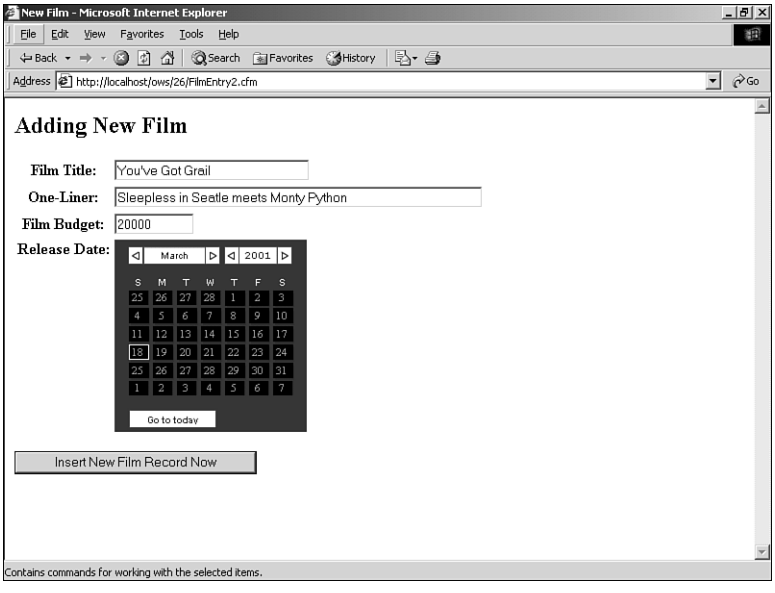

Figure 26.2
The Calendar component can also be used in an inline mode.

The `ONSUBMIT` attribute of the `<CFFORM>` tag causes the JavaScript `checkForm()` function to be called whenever the user attempts to submit the form. If a date value has not been

selected yet, the function will display a "required" type of message to the user and return a value of `false`, which is what causes the form submission to be blocked. If the date value has been provided, the function returns `true`, which enables the form to be submitted.

USING THE CALCULATOR COMPONENT

The Calculator component is similar conceptually to the Calendar. It behaves in either a pop-up or an inline mode (controlled by the `POPUP` attribute) and is well suited for augmenting a normal data-entry form.

To include the Calculator in one of your pages, use the `<CF_uicalculator>` tag. Table 26.3 lists the attributes this tag supports.

TABLE 26.3 `<CF_uicalculator>` TAG ATTRIBUTES

Attribute	Description
`FORMFIELD`	A name for the component's value, which will be available to the form's `ACTION` page when the form is submitted. This attribute is equivalent to the `NAME` attribute of an ordinary form control, such as `<INPUT>`, `<TEXTAREA>`, or `<CFINPUT>`.
`POPUP`	`Yes` or `No`. If `Yes`, which is the default, the user has an ordinary text field for entering the date. The text field has a small calculator icon next to it, which the user can use to invoke the calculator. If `No`, the calculator appears right away, without any separate text entry field next to it.
`WIDTH`	The width of the calculator, in pixels.
`HEIGHT`	The height of the calculator, in pixels.
`stFORMATTING`	A ColdFusion structure containing the generic formatting properties supported by the Flash Component Kit. Refer to Listing 26.3 for details.
`ZINDEX`	A `zindex` value for the Calculator component, which can be used to control whether the calculator appears in front of or behind other dynamic elements on the page. If not provided, a default value of `100` is used; use a higher number if the calculator appears behind other absolutely positioned elements on the page. Currently relevant only when the Calculator is viewed with Internet Explorer. For more information about `zindex` and absolutely positioned elements, refer to a DHTML reference book or the DHTML References section of `http://msdn.microsoft.com/`.
`REQUIRED`	Undocumented. `Yes` or `No`. If `Yes`, the user must provide a value before the form can be submitted. Corresponds to the `REQUIRED` attribute of the `<CFINPUT>` tag. This attribute is relevant only when `POPUP="Yes"`. It has no effect if `POPUP="No"`.
`MESSAGE`	Undocumented. The message to be shown if `REQUIRED="Yes"` and the user leaves the value blank. Corresponds to the `MESSAGE` attribute of the `<CFINPUT>` tag. If omitted, a default message (`Error in numeric data entry field`) is used. This attribute is relevant only when `POPUP` and `REQUIRED` are both `Yes`.

Listing 26.5 is a revised version of the `FilmEntry.cfm` template from Listing 26.2. Now, in addition to being able to use the calendar icon to provide a date, the user can use the calculator icon to provide the film's budget. The user can use the Calculator component to perform any estimations or calculations to help come up with an accurate estimate of the film's final cost. Most functions available in a traditional handheld calculator are available here, including square roots, memory recall, and so on. When finished calculating, the user just clicks the OK button on the Calculator, which places the final calculated value onto the form, as shown in Figure 26.3.

LISTING 26.5 `FilmEntry3.cfm`—ADDING THE CALCULATOR COMPONENT TO A FORM

```
<HTML>
<HEAD><TITLE>New Film</TITLE></HEAD>
<BODY>
<H2>Adding New Film</H2>

<!--- If User is Submitting the Form --->
<CFIF IsDefined("FORM.MovieTitle")>
  <!--- Insert new record into Films table in database --->
  <CFINSERT
    DATASOURCE="#REQUEST.DataSource#"
    TABLENAME="Films">

  <!--- Display success message --->
  <CFOUTPUT>
    <B>#FORM.MovieTitle#</B> was added.<BR>
  </CFOUTPUT>
</CFIF>

<!--- Self-submitting data-entry form --->
<CFFORM ACTION="#CGI.SCRIPT_NAME#" METHOD="POST">
  <TABLE>
    <!--- Text entry field for: MovieTitle --->
    <TR>
      <TH>Film Title:</TH>
      <TD>
        <CFINPUT
          NAME="MovieTitle"
          SIZE="30"
          MAXLENGTH="50"
          REQUIRED="Yes"
          MESSAGE="You may not leave the title blank.">
      </TD>
    </TR>

    <!--- Text entry field for: MovieTitle --->
    <TR>
      <TH>One-Liner:</TH>
      <TD>
        <CFINPUT
          NAME="PitchText"
          SIZE="60"
          MAXLENGTH="200"
          REQUIRED="Yes"
          MESSAGE="You may not leave the One-Liner blank.">
```

LISTING 26.5 CONTINUED

```
      </TD>
    </TR>

    <!--- Text entry field for: AmountBudgeted --->
    <!--- Augmented with Flash Calendar component from FCK --->
    <TR>
      <TH>Film Budget:</TH>
      <TD>
        <CF_uiCALCULATOR
          FORMFIELD="AmountBudgeted"
          WIDTH="200"
          HEIGHT="200"
          POPUP="Yes"
          stFORMATTING="#REQUEST.uiToolkitFormatting#"
          REQUIRED="Yes"
          MESSAGE="You must provide the budget for the film.">
      </TD>
    </TR>

    <!--- Text entry field for: DateInTheaters --->
    <!--- Augmented with Flash Calendar component from FCK --->
    <TR>
      <TH>Release Date:</TH>
      <TD>
        <CF_uicalendar
          FORMFIELD="DateInTheaters"
          WIDTH="200"
          HEIGHT="200"
          POPUP="Yes"
          SELBDRCOLOR="##FFFF00"
          stFORMATTING="#REQUEST.uiToolkitFormatting#"
          REQUIRED="Yes"
          MESSAGE="You must provide a release date.">
      </TD>
    </TR>

  </TABLE>

  <!--- Submit Button to create new film record --->
  <P><INPUT TYPE="Submit" VALUE="Insert New Film Record Now">
</CFFORM>

</BODY>
</HTML>
```

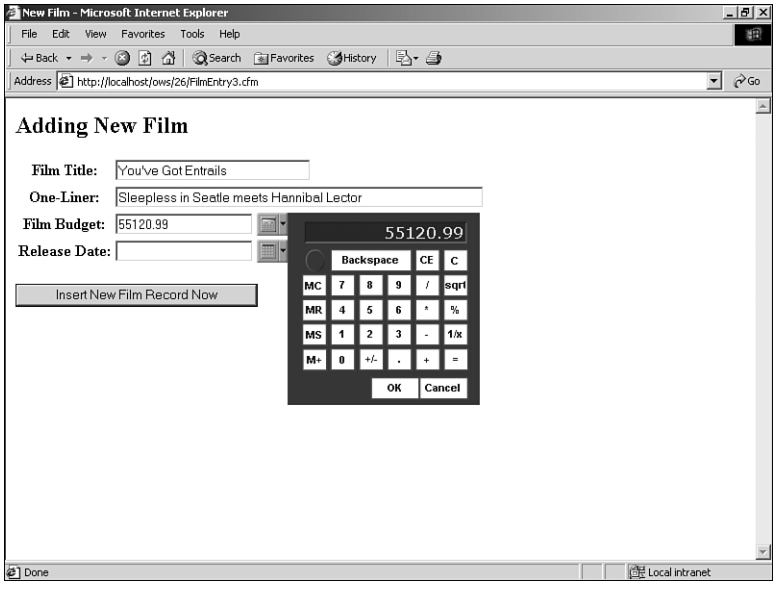

Figure 26.3
The Calculator provides a terrific way for users to make quick computations while filling out a form.

USING THE NAVIGATION BAR COMPONENT

The Navigation Bar component is used to provide an interactive navigation bar for your application pages. The Navigation Bar supports two levels of navigation choices. The top-level choices are always visible. The second-level choices become visible when the user selects the corresponding top-level choice, creating a pull-down-menu effect.

Because of the way most browsers work with plug-ins such as the Macromedia Flash Player, the Navigation Bar must always take up the same amount of space onscreen, regardless of whether any of the top-level choices are currently selected. If the second-level choices are too numerous to display at once within the space you specify with the `WIDTH` and `HEIGHT` attributes, arrow elements will be provided to enable the user to scroll through the second-level choices.

INTRODUCING THE NAVIGATION BAR TAGS

To include the Navigation Bar component in your pages, use the `<CF_uihnavbar>` tag. Then, within the `<CF_uihnavbar>` tag, add a `<CF_uihnavbaroption>` tag for each top-level navigation choice. Finally, within each `<CF_uihnavbaroption>` tag, add a `<CF_uihnavbaroptionitem>` tag for each second-level choice.

Tables 26.4, 26.5, and 26.6 list the attributes supported by each of the Navigation Bar tags.

TABLE 26.4 <CF_uihnavbar> TAG ATTRIBUTES

Attribute	Description
WIDTH	The width of the navigation bar, in pixels.
HEIGHT	The height of the navigation bar, in pixels.
stFORMATTING	A ColdFusion structure containing the generic formatting properties supported by the Flash Component Kit. Refer to Listing 26.3 for details.
DROPSPEED	A number between 0 and 100 that indicates how quickly the second-level elements should drop down. The higher the number, the faster the elements appear. The default value of 0 is far too slow for most situations; we recommend experimenting with an initial value of 99. Use 100 for an instant drop-down effect.
SUBSTEP	A number between 0 and 100 that indicates how quickly the user will scroll through the second-level elements, if there are too many elements to show in the navigation bar at once. We recommend experimenting with an initial value of 25.
TOPHEIGHT	The height, in pixels, of the top portion of the navigation bar (where the top-level choices appear). For best results, the TOPHEIGHT and BOTTOMHEIGHT should add up to the overall HEIGHT.
BOTTOMHEIGHT	The height, in pixels, of the bottom portion of the navigation bar (where the second-level choices appear). For best results, the TOPHEIGHT and BOTTOMHEIGHT should add up to the overall HEIGHT.
MIDLINECOLOR	The color of the line that separates the top and bottom portions of the navigation bar, as an RGB hexadecimal value.
ARROWCOLOR	The color of the arrow icon that appears when the user needs to scroll through the second-level choices, as an RGB hexadecimal value.
ARROWBGCOLOR	The background color that surrounds the arrow icon that appears when the user needs to scroll through the second-level choices, as an RGB hexadecimal value.

TABLE 26.5 <CF_uihnavbaroption> TAG ATTRIBUTES

Attribute	Description
NAME	The text that should appear for the top-level navigation choice.
WIDTH	The width of the top-level item, in pixels. If it's not provided, a default width of 50 pixels is used.

TABLE 26.6 <CF_uihnavbaroptionitem> TAG ATTRIBUTES

Attribute	Description
NAME	Required. The text that should appear for the top-level navigation choice.
URL	Required. The URL the user should be sent to if he clicks the item.
WIDTH	The width of the second-level navigation item, in pixels. If it's not provided, a default width of 50 pixels is used.

USING THE NAVIGATION BAR IN A PAGE HEADER

Listing 26.6 demonstrates one way the Horizontal Navigation Bar component can be used. It creates an interactive pull-down header that can be placed at the top of any page via a `<CFINCLUDE>` tag.

Using the navigation bar, the user can navigate to the Orange Whip Studios home page or to the online store. Users also can scroll through lists of current films, actors, and actresses; when they click one of these menu items, they are sent to the appropriate page, with the appropriate `FilmID` or `ActorID` passed as a URL parameter.

LISTING 26.6 NavBarHeader.cfm—USING THE HORIZONTAL NAVIGATION BAR COMPONENT

```
<!--- Fetch a listing of current films from the database --->
<CFQUERY NAME="GetFilms" DATASOURCE="#REQUEST.DataSource#"
  CACHEDWITHIN="#CreateTimeSpan(0,0,10,0)#">
  SELECT FilmID, MovieTitle
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- Fetch a listing of current actors from the database --->
<CFQUERY NAME="GetActors" DATASOURCE="#REQUEST.DataSource#"
  CACHEDWITHIN="#CreateTimeSpan(0,0,10,0)#">
  SELECT ActorID, NameFirst, NameLast, Gender
  FROM Actors
  WHERE IsTotalBabe = 1
</CFQUERY>

<!--- Use Query-of-queries to extract just the female actors --->
<CFQUERY NAME="GetFemaleActors" DBTYPE="Query">
  SELECT ActorID, NameFirst, NameLast
  FROM GetActors
  WHERE Gender = 'F'
  ORDER BY NameLast, NameFirst
</CFQUERY>

<!--- Use Query-of-queries to extract just the male actors --->
<CFQUERY NAME="GetMaleActors" DBTYPE="Query">
  SELECT ActorID, NameFirst, NameLast
  FROM GetActors
  WHERE Gender = 'M'
  ORDER BY NameLast, NameFirst
</CFQUERY>

<!--- Create formatting structure --->
<CFSET Formatting = StructNew()>
<CFSET Formatting.bgcolor = "##F5DEB3">
<CFSET Formatting.btncolor = "##FF8C00">
<CFSET Formatting.btnhicolor = "##FFA500">

<TABLE>
<TR>
  <TD>
    <IMG SRC="../images/logo_c.gif" WIDTH="101" HEIGHT="101" ALT="" BORDER="0">
  </TD>
```

LISTING 26.6 CONTINUED

```
<TD>
  <!--- Include Navigation Bar component, from Flash Component Kit --->
  <CF_uihnavbar
    WIDTH="500"
    HEIGHT="70"
    DROPSPEED="99"
    SUBSTEP="25"
    ARROWCOLOR="##FFFF00"
    ARROWBGCOLOR="##FF0000"
    BGCOLOR="##FF0000"
    TOPHEIGHT="25"
    BOTTOMHEIGHT="45"
    stFORMATTING="#Formatting#">

    <!--- Include Top-Level navigation choice --->
    <CF_uihnavbaroption NAME="Orange Whip Studios" WIDTH="200">
      <!--- Include Second-Level navigation choice --->
      <CF_uihnavbaroptionitem
        NAME="Home"
        URL="../"
        WIDTH="100">
      <!--- Include Second-Level navigation choice --->
      <CF_uihnavbaroptionitem
        NAME="Store"
        URL="../29/Store.cfm"
        WIDTH="100">
    </CF_uihnavbaroption>


    <!--- Include Top-Level navigation choice for Films --->
    <CF_uihnavbaroption NAME="Films" WIDTH="100">
      <!--- For each Film --->
      <CFLOOP QUERY="GetFilms">
        <!--- URL to send user to if they click on this film --->
        <CFSET LinkURL = "ShowFilm.cfm?FilmID=#GetFilms.FilmID#">
        <!--- Calculate appropriate width, based on length of film title --->
        <CFSET LinkWidth = 30 + (Len(GetFilms.MovieTitle) * 6)>
        <!--- Include Second-Level navigation choice for this film --->
        <CF_uihnavbaroptionitem
          NAME="#GetFilms.MovieTitle#"
          URL="#LinkURL#"
          WIDTH="#LinkWidth#">
      </CFLOOP>
    </CF_uihnavbaroption>

    <!--- Include Top-Level navigation choice for Actors (Female) --->
    <CF_uihnavbaroption NAME="Hot Actresses" WIDTH="100">
      <!--- For each Film --->
      <CFLOOP QUERY="GetFemaleActors">
        <CFSET FullName = NameFirst & " " & NameLast>
        <!--- URL to send user to if they click on this film --->
        <CFSET LinkURL = "ShowActor.cfm?ActorID=#GetFemaleActors.ActorID#">
        <!--- Calculate appropriate width, based on length of FullName --->
        <CFSET LinkWidth = 30 + (Len(FullName) * 6)>
        <!--- Include Second-Level navigation choice for this Actor --->
```

```
        <CF_uihnavbaroptionitem
          NAME="#FullName#"
          URL="#LinkURL#"
          WIDTH="#LinkWidth#">
      </CFLOOP>
    </CF_uihnavbaroption>

    <!--- Include Top-Level navigation choice for Actors (Male) --->
    <CF_uihnavbaroption NAME="Hot Actors" WIDTH="100">
      <!--- For each Film --->
      <CFLOOP QUERY="GetMaleActors">
        <CFSET FullName = NameFirst & " " & NameLast>
        <!--- URL to send user to if they click on this film --->
        <CFSET LinkURL = "ShowActor.cfm?ActorID=#GetMaleActors.ActorID#">
        <!--- Calculate appropriate width, based on length of FullName --->
        <CFSET LinkWidth = 30 + (Len(FullName) * 6)>
        <!--- Include Second-Level navigation choice for this Actor --->
        <CF_uihnavbaroptionitem
          NAME="#FullName#"
          URL="#LinkURL#"
          TXTCOLOR="##FF0000"
          WIDTH="#LinkWidth#">
      </CFLOOP>
    </CF_uihnavbaroption>

  </CF_uihnavbar>

 </TD>
</TR>
</TABLE>
```

This template uses four `<CFQUERY>` tags. The first query gets a listing of all films currently in the `Films` table. The second query gets a listing of all `Actors` in the Actors table. Per a new company-wide mandate from the marketing department, only the really attractive actors are retrieved. (It seems that next month's ad campaign goes something like "Only Hotties On This Web Site!") Then, two query-of-queries are run, which split the records for attractive men and attractive women into two separate queries called `GetFemaleActors` and `GetMaleActors`.

Next, the `<CF_uihnavbar>` tag is used to create the navigation bar. Within the navigation bar, `<CF_uihnavbaroption>` is used to create a top-level navigation choice titled Orange Whip Studios. Within this top-level choice, two `<CF_uihnavbaroptionitem>` tags are used to provide links to the company's home page and the online store template from Chapter 29, "Online Commerce."

Another `<CF_uihnavbaroption>` is then used to create another top-level navigation choice, called Films. Within this top-level choice, a `<CF_uihnavbaroptionitem>` is generated for each film in the `GetFilms` query. Because the movie titles are of differing lengths, a simple calculation is performed to come up with an appropriate width for each film's menu item: All menu items start at a base width of 30 pixels, plus an additional 8 pixels for each letter in the movie title. Because the font being used is not monospaced, this is just an approximate value, but it works quite well for the film titles in the database.

A similar loop is used to create top-level options named Actresses and Actors, including a menu item for each record returned by the `GetFemaleActors` and `GetMaleActors` queries. The result is an attractive menu that includes four top-level choices and however many second-level choices are appropriate, based on the number of film and actor records currently in the database.

Listing 26.7 places the pull-down header at the top of a hypothetical home page with a `<CFINCLUDE>` tag. Figure 26.4 shows the results when this template is viewed in a Web browser. You could add this `<CFINCLUDE>` tag to any template on which you want the navigation header to appear.

Tip

If you want the navigation header to appear at the top of each of your application's pages, you could simply add the `<CFINCLUDE>` tag to your `Application.cfm` file.

LISTING 26.7 `HomePage1.cfm`—INCLUDING THE NAVIGATION BAR HEADER AT THE TOP OF A PAGE

```
<HTML>
<HEAD><TITLE>Hypothetical Home Page</TITLE></HEAD>
<BODY>

<!--- Flash-Based Navigation Bar --->
<CFINCLUDE TEMPLATE="NavBarHeader.cfm">

<P>This is the home page of Orange Whip Studios.
We hope you enjoy your visit to our web site.<BR>

<P>You can be assured that we at Orange Whip Studios have exacting standards
for our films.  We bring you only the best in entertainment.
For instance, all of our actors and actresses are especially attractive.
Also, we mix the soundtracks and voice-overs especially loud in our historical
dramas and WWII films, like Girl Harbor (coming soon).

</BODY>
</HTML>
```

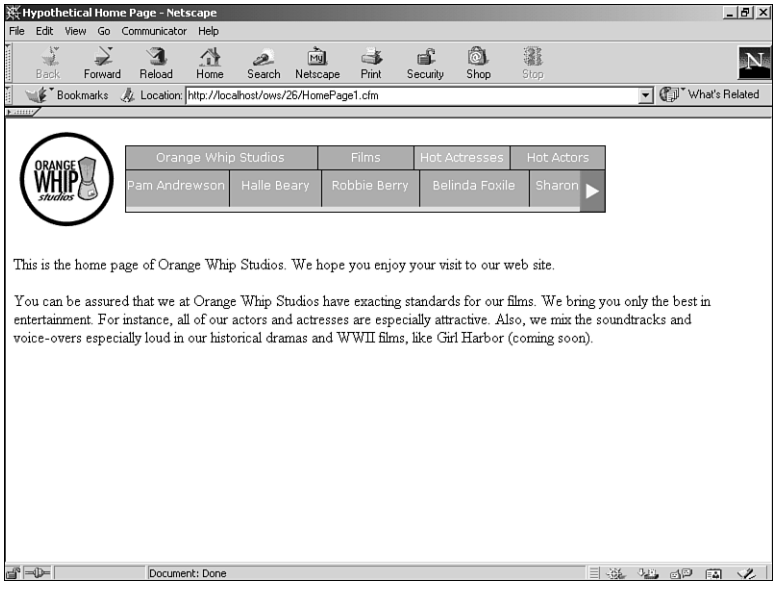

Figure 26.4
The Horizontal Navigation Bar component provides a slick, interactive way for users to navigate your site.

USING THE IE CASCADING MENU COMPONENT

In addition to the Horizontal Navigation Bar component, the Component Kit also provides an interactive, expanding Cascading Menu component. This component is similar to the Horizontal Navigation Bar, but it can expand over the Web page as necessary (somewhat similar to the Windows Start menu), creating a more pleasant experience for your users.

To add the IE Cascading Menu component to one of your application pages, use the `<CF_uicascadingmenu>` and `<CF_uicascadingmenuitem>` tags. The attributes supported by these tags are listed in Tables 26.7 and 26.8.

Note

The Cascading Menu component works only with Internet Explorer and doesn't work on the Macintosh. See Listing 26.9 for an example of how to show the appropriate component according to each user's browser and platform.

TABLE 26.7 `<CF_uicascadingmenu>` TAG ATTRIBUTES

Attribute	Description
`WIDTH`	The width, in pixels, of the Cascading Menu component.
`HEIGHT`	The height, in pixels, of the Cascading Menu component. This does not affect how tall the menu is when it first appears. It just puts a boundary on how large the menu can get as it expands to display the items the user pulls down from the initial menu items.
`stFORMATTING`	A ColdFusion structure containing the generic formatting properties supported by the Flash Component Kit. Refer to Listing 26.3 for details.

TABLE 26.7 CONTINUED

Attribute	Description
SCALE	A relative measure that affects the size of the individual items in the menu. The default is 1. A smaller number makes the items smaller, and a larger number makes them larger. The maximum number is 10. You can specify nonintegers, such as 1.3 or 0.9.
ELASTICITY	A number from 1 to 100 that governs how much the top-level items expand when selected. The lower the number, the more they will expand. We recommend experimenting with an initial value of 30.
BOOST	A number that indicates how fast the top-level items expand when selected. A 1 indicates an almost nonanimated expansion, and 100 indicates a very animated expansion. We recommend experimenting with an initial value of 20.
DAMPING	A number that affects the bouncing effect when top-level items are selected. 100 indicates no bouncing, and 1 causes the bouncing to go on for a long time. We recommend experimenting with an initial value of 50.
TRANSPARENCY	A transparency measure for the menu, between 0 and 100.

TABLE 26.8 <CF_uicascadingmenuitem> TAG ATTRIBUTES

Attribute	Description
ID	A numeric identifier for this menu item. This should be unique within the cascading menu.
PARENTID	The parent item of this menu item. If it's provided, this menu item becomes a child of (appears beneath or beside) the parent. If it's not provided, the menu item becomes a top-level item that is visible when the menu first appears on the page.
NAME	The text to display in the menu item.
HREF	The URL the user should be brought to if she clicks the menu item.
TARGET	The target frame, if any, that should be used when the user clicks the menu item. Corresponds to the TARGET attribute of the A tag. For details about frames and targets, see Chapter 11, "Creating Data-Driven Pages."
BGCOLOR	The background color for the menu item, as a hexadecimal RGB color. If not provided, the color is inherited from the parent item or from the value specified by the formatting structure provided to the stFORMATTING attribute for the Cascading Menu as a whole.
BGHICOLOR	The background color to use when the item is moused over. If not provided, it is inherited (see BGCOLOR).
TXTCOLOR	The text color for the menu item. If not provided, it is inherited (see BGCOLOR).
TXTHICOLOR	The text color to use when the item is moused over. If not provided, it is inherited (see BGCOLOR).
SUBBGCOLOR	The background color for this item's children, if any.

Attribute	Description
SUBBGHICOLOR	The mouse-over color for this item's children, if any.
SUBTXTCOLOR	The text color for this item's children, if any.
SUBTXTHICOLOR	The mouse-over text color for this item's children, if any.

USING THE CASCADING MENU IN A PAGE HEADER

Listing 26.8 shows how the Cascading Menu component can be put to work in your application pages. Here, it is used to provide the same set of menu choices that was presented with the Navigation Bar component (refer to Listing 26.6). Instead of having to scroll through the second-level options, the user can see all the items at once, in a somewhat more intuitive pull-down format. The results are shown in Figure 26.5.

LISTING 26.8 `CascadingHeader.cfm`—USING THE CASCADING MENU COMPONENT

```
<!--- Fetch a listing of current films from the database --->
<CFQUERY NAME="GetFilms" DATASOURCE="#REQUEST.DataSource#"
  CACHEDWITHIN="#CreateTimeSpan(0,0,10,0)#">
  SELECT FilmID, MovieTitle
  FROM Films
  ORDER BY MovieTitle
</CFQUERY>

<!--- Fetch a listing of current actors from the database --->
<CFQUERY NAME="GetActors" DATASOURCE="#REQUEST.DataSource#"
  CACHEDWITHIN="#CreateTimeSpan(0,0,10,0)#">
  SELECT ActorID, NameFirst, NameLast, Gender
  FROM Actors
  WHERE IsTotalBabe = 1
</CFQUERY>

<!--- Use Query-of-queries to extract just the female actors --->
<CFQUERY NAME="GetFemaleActors" DBTYPE="Query">
  SELECT ActorID, NameFirst, NameLast
  FROM GetActors
  WHERE Gender = 'F'
  ORDER BY NameLast, NameFirst
</CFQUERY>

<!--- Use Query-of-queries to extract just the male actors --->
<CFQUERY NAME="GetMaleActors" DBTYPE="Query">
  SELECT ActorID, NameFirst, NameLast
  FROM GetActors
  WHERE Gender = 'M'
  ORDER BY NameLast, NameFirst
</CFQUERY>

<!--- Create formatting structure --->
<CFSET Formatting = StructNew()>
<CFSET Formatting.bgcolor = "##F5DEB3">
<CFSET Formatting.btncolor = "##FF8C00">
<CFSET Formatting.btnhicolor = "##FFA500">
```

LISTING 26.8 CONTINUED

```
<TABLE>
<TR VALIGN="top">
  <TD>
    <IMG SRC="../images/logo_c.gif" WIDTH="101" HEIGHT="101" ALT="" BORDER="0">
  </TD>
  <TD>
    <!--- Include Navigation Bar component, from Flash Component Kit --->
    <CF_uicascadingmenu
      WIDTH="500"
      HEIGHT="300"
      ELASTICITY="30"
      BOOST="20"
      SCALE="1"
      DAMPING="50"
      stFORMATTING="#Formatting#">

      <!--- Include Top-Level navigation choice --->
      <CF_uicascadingmenuitem
        ID="1"
        NAME="Orange Whip Studios">
      <!--- Include Second-Level navigation choice --->
      <CF_uicascadingmenuitem
        NAME="Home"
        ID="2"
        PARENTID="1"
        HREF="../">
      <!--- Include Second-Level navigation choice --->
      <CF_uicascadingmenuitem
        NAME="Store"
        ID="3"
        PARENTID="1"
        HREF="../29/Store.cfm">


      <!--- Include Top-Level navigation choice for Films --->
      <CF_uicascadingmenuitem
        ID="4"
        BGCOLOR="##FF0000"
        NAME="Films">
      <!--- Include sub-navigation choice for A-M films --->
      <CF_uicascadingmenuitem
        ID="1000"
        PARENTID="4"
        NAME="A-L">
      <!--- Include sub-navigation choice for N-Z films --->
      <CF_uicascadingmenuitem
        ID="1500"
        PARENTID="4"
        NAME="M-Z">

      <!--- For each Film --->
      <CFLOOP QUERY="GetFilms">
        <!--- URL to send user to if they click on this film --->
        <CFSET LinkURL = "ShowFilm.cfm?FilmID=#GetFilms.FilmID#">
        <!--- ID number for this menu item (1000 plus current row) --->
```

```
        <CFSET ParentID = IIF(Left(MovieTitle, 1) LTE "M", 1000, 1500)>
        <CFSET LinkID = ParentID + GetFilms.CurrentRow>

        <!--- Include Second-Level navigation choice for this film --->
        <CF_uicascadingmenuitem
          ID="#LinkID#"
          PARENTID="#ParentID#"
          NAME="#GetFilms.MovieTitle#"
          HREF="#LinkURL#">
      </CFLOOP>


      <!--- Include Top-Level navigation choice for Actors (Female) --->
      <CF_uicascadingmenuitem
        ID="2000"
        NAME="Hot Actresses" WIDTH="100">
      <!--- For each Film --->
      <CFLOOP QUERY="GetFemaleActors">
        <CFSET FullName = NameFirst & " " & NameLast>
        <!--- URL to send user to if they click on this actor --->
        <CFSET LinkURL = "ShowActor.cfm?ActorID=#GetFemaleActors.ActorID#">
        <!--- ID number for this menu item (1000 plus current row) --->
        <CFSET LinkID = 1000 + GetFilms.CurrentRow>
        <!--- Include Second-Level navigation choice for this Actor --->
        <CF_uicascadingmenuitem
          ID="#LinkID#"
          PARENTID="2000"
          NAME="#FullName#"
          HREF="#LinkURL#">
      </CFLOOP>

      <!--- Include Top-Level navigation choice for Actors (Female) --->
      <CF_uicascadingmenuitem
        ID="3000"
        NAME="Hot Actors" WIDTH="100">
      <!--- For each Film --->
      <CFLOOP QUERY="GetMaleActors">
        <CFSET FullName = NameFirst & " " & NameLast>
        <!--- URL to send user to if they click on this actor --->
        <CFSET LinkURL = "ShowActor.cfm?ActorID=#GetMaleActors.ActorID#">
        <!--- ID number for this menu item (1000 plus current row) --->
        <CFSET LinkID = 3000 + GetFilms.CurrentRow>
        <!--- Include Second-Level navigation choice for this Actor --->
        <CF_uicascadingmenuitem
          ID="#LinkID#"
          PARENTID="3000"
          NAME="#FullName#"
          HREF="#LinkURL#">
      </CFLOOP>

    </CF_uicascadingmenu>

  </TD>
</TR>
</TABLE>
```

This listing uses the same queries and basic looping strategy that was used in the Navigation Bar header (refer to Listing 26.6). The main difference between the two versions is the need to have unique numbers to provide to the `ID` attribute for each `<CF_uicascadingmenuitem>` tag.

The simple technique used in this template is to give each of the static menu items (the ones marked Films, Hot Actors, and so on) a hard-coded ID number. For each of the dynamically generated menu items, an ID number is created by adding the current query row number to the parent item's ID number. For instance, the ID number for the Hot Actresses menu item is `2000`. So, within the `<CFLOOP>` for the `GetFemaleActors` query, a `LinkID` is created for each actress's menu item by adding the value of `GetFemaleActors.CurrentRow` to `2000`. The first actress's menu item will have an ID of `2001`, the next will have an ID of `2002`, and so on.

As long as there aren't more than 1,000 actresses, this technique will work fine (and if there were more than 1,000, it probably wouldn't make sense to display them in a cascading menu like this).

DISPLAYING THE CASCADING MENU ONLY FOR IE

Because the Cascading Menu component works only if Internet Explorer is being used (and not on the Macintosh platform), it makes sense to add a bit of code that detects which browser the user is using to visit the page.

Listing 26.9 adds a simple `<CFIF>` test to the `HomePage.cfm` file, which displays the new Cascading Menu version of the page header if the user is visiting the page with IE on a Windows machine, as shown in Figure 26.5. If not, the Navigation Bar version of the page header from Listing 26.6 is displayed (refer to Figure 26.4).

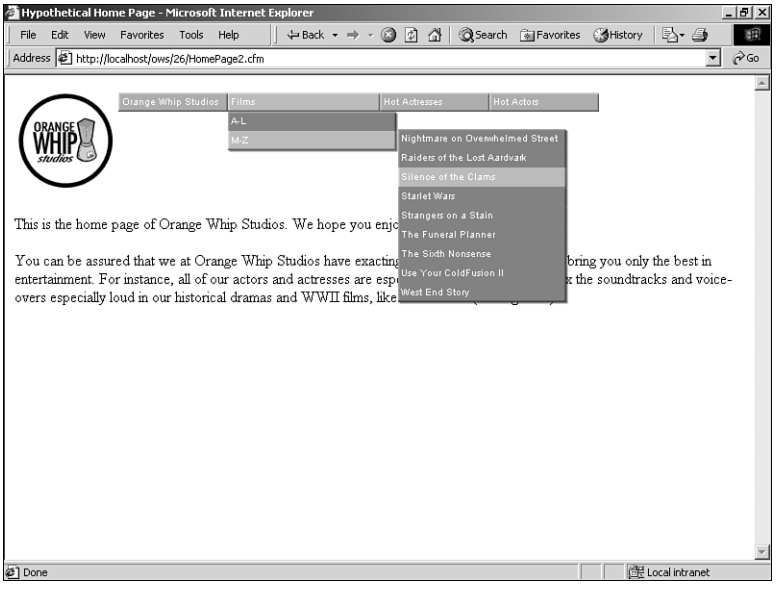

Figure 26.5
The Cascading Menu component provides an interactive menu that expands as the user drills down into it.

LISTING 26.9 HomePage2.cfm—USING THE CASCADING OR HORIZONTAL VERSION OF THE PAGE HEADER, DEPENDING ON THE BROWSER

```
<HTML>
<HEAD><TITLE>Hypothetical Home Page</TITLE></HEAD>
<BODY>

<!--- Flash-Based Navigation Bar --->
<CFIF (CGI.HTTP_USER_AGENT contains "MSIE")
  AND (CGI.HTTP_USER_AGENT contains "Windows")>
  <CFINCLUDE TEMPLATE="CascadingHeader.cfm">
<CFELSE>
  <CFINCLUDE TEMPLATE="NavBarHeader.cfm">
</CFIF>

<P>This is the home page of Orange Whip Studios.
We hope you enjoy your visit to our web site.<BR>

<P>You can be assured that we at Orange Whip Studios have exacting standards
for our films.  We bring you only the best in entertainment.
For instance, all of our actors and actresses are especially attractive.
Also, we mix the soundtracks and voice-overs especially loud in our historical
dramas and WWII films, like Girl Harbor (coming soon).

</BODY>
</HTML>
```

Note

For more information about the `CGI.HTTP_USER_AGENT` variable used here to do the browser detection, see Appendix C, "Special ColdFusion Variables and Result Codes."

USING YOUR OWN FLASH MOVIES IN COLDFUSION PAGES

So far, this chapter has focused on how to use the prebuilt interactive widgets Macromedia has made available in the Flash Component Kit. Of course, you are also free to create your own Flash movies for inclusion in your pages. These movies might be navigational elements, data-collection widgets, or just pretty animations that contribute to the overall look and feel of your application.

To create Flash movies, you use the program called Macromedia Flash, which is sold separately from ColdFusion. A free 30-day trial version of Macromedia Flash 5 has been provided on the CD-ROM for this book and is also available from the Downloads section at `http://www.macromedia.com`.

Unfortunately, it just isn't possible for us to teach you how to create Flash movies in this chapter. If you would like to learn how to create your own Flash movies, please consult a book about Flash 5 or take a look at the tutorial provided in Flash 5 itself. The movies for the examples in this chapter have been provided on the CD-ROM for this chapter.

Note

Don't confuse Macromedia Flash with the Macromedia Flash *Player*. Macromedia Flash is what you use to create Flash movies and must be purchased separately. The Flash Player is what's used to view Flash movies in a browser and is available for free download. The Player is also automatically installed with most recent Web browsers, so most users already have the Player installed on their machines.

Note

You can find out the percentage of users who currently have the Flash Player installed on their machines by visiting `www.macromedia.com/software/player_census`.

FLASH MOVIE CONCEPTS

Macromedia Flash movies provide a method to supply compact, vector-based animations and other interactive elements for your Web pages. Macromedia Flash movies generally work identically across browsers and platforms and can even be displayed on devices such as WebTV, PocketPC handhelds, and soon on the Sony PlayStation 2.

Flash movies can collect information from users via form-like interfaces, be printed, be zoomed in on, and generally provide an exciting experience for your users. Best of all, you can create Flash movies that do a great deal but that are extremely small in terms of file size, so your users will not have to wait for a long time to view them. For instance, all the sample movies used in this chapter are less than 10KB. The Flash Player can even begin displaying a movie before the entire file has been downloaded.

In other words, Flash is a terrific way to present interactive elements to your users, without the programming hassles associated with Dynamic HTML or client-side Java.

FLASH FILE TYPES

The three types of Flash-related files that you are likely to encounter are as follows:

- **`.fla` files**—While you are creating a movie in the Macromedia Flash authoring environment, your work is saved as an `.fla` file. This file can't be displayed on a Web page by the Flash Player. The Flash environment refers to these `.fla` files as *Flash Movie* files, which is somewhat confusing. Just think of `.fla` files as the author-able version of the final Flash presentation. You would share this file with others only if you wanted them to be able to edit the movie.
- **`.swf` files**—These are the files the Flash Player can display on a Web page. An `.swf` file is the unchangeable, compiled, compressed output generated by the corresponding `.fla` file. For instance, if you were creating an animation for Orange Whip Studios, you might be working with a file called `owsAnimation.fla`. When finished, you would tell Flash to publish the `owsAnimation.swf` file, which you could then display on your application's pages. The Flash IDE refers to these `.swf` files as *Flash Player* files.
- **`.swt` files**—These are *Generator templates*, which are used by the Macromedia Generator product (sold separately) to produce customized versions of movies on the

server. Conceptually, a Generator template is similar to a ColdFusion template, except that the output is a Flash Player file (.swf) instead of HTML. If you are curious about Generator templates, you are encouraged to download a trial version of Macromedia Generator from http://www.macromedia.com.

Note

If you are familiar with Java, think of .fla files as being conceptually equivalent to .java files (they are both source material that has not yet been compiled for delivery), and .swf files as being similar to the resulting .class files (they are both pseudocompiled files that need to be interpreted at runtime). Continuing the analogy, you can thus think of the Macromedia Flash Player as the rough equivalent to the Java Virtual Machine (both provide platform-specific host environments for miniature, platform-agnostic programs), and the Macromedia Flash IDE as being equivalent to your Java IDE or compiler.

INCLUDING FLASH MOVIES IN YOUR PAGES

For the purposes of this chapter, pretend that you already have a Flash Player (.swf) file you want to incorporate into your ColdFusion application. Perhaps you created the movie yourself with the Flash IDE, or perhaps a graphic artist supplied it to you. In any case, to include the movie in a page, you must use <OBJECT>, <PARAM>, and <EMBED> tags.

Note

The <OBJECT> and <PARAM> tags are understood by Internet Explorer, and <EMBED> is used to specify the same properties for Netscape browsers. For the movie to be displayed correctly in all browsers, you must use the tags together as shown in this section.

Table 26.9 explains the special properties you can use to control how the Flash movie is displayed. Most of these properties are supplied twice—once to the <EMBED> tag and once for the <OBJECT> and <PARAM> tags. You see how to use these properties in Listing 26.10, later in this chapter.

TABLE 26.9 IMPORTANT PROPERTIES TO USE WHEN DISPLAYING A FLASH MOVIE WITH <OBJECT> AND <EMBED>

Property	Description
SRC and MOVIE	The relative URL path to the Flash Player (.swf) file you want to display. Similar conceptually to the SRC attribute of the <IMG> tag you are already familiar with. In your code, you should provide SRC for the <EMBED> tag and MOVIE for the <OBJECT> tag, using the same path value for both.
WIDTH	The width at which you want to show the movie, in pixels (such as WIDTH="50") or as a percentage (such as WIDTH="100%"). Flash movies are scalable, so you typically can display a movie at whatever width you choose, regardless of how big it was when it was created.

TABLE 26.9 CONTINUED

Property	Description
HEIGHT	The height at which you want to show the movie, in pixels or as a percentage.
PLAY	Whether the movie should be playing when the page first appears. If `true` (the default), the movie starts playing on its own. If `false`, the movie appears stopped at the first frame until the user starts it via the right-click menu (or it is started programmatically via some type of script).
LOOP	tag;properties;LOOP>Whether the movie repeats indefinitely or stops when it reaches the last frame. If `true` (the default), the movie repeats indefinitely. If `false`, it stops at the last frame. Note that the movie's author is free to override this behavior by adding ActionScript to the movie during its creation.
MENU	Whether a pop-up menu displays when the user right-clicks the movie area. If `true` (the default), the default menu appears, which provides the user with options to print the movie, zoom in or out, start and stop the movie, and so on. If `false`, the pop-up menu contains only an About Macromedia Flash Player option.
QUALITY	Can be set to `low`, `high`, `autolow`, `autohigh`, or `best`. The default value of `high` is usually the most appropriate. See the Flash documentation for details.
BGCOLOR	Specifies an optional background color for the movie, which overrides the background color applied when the movie was created.
WMODE	Whether the movies displays transparently or as a background layer. If set to `Window` (the default), the movie displays normally. `Opaque` makes the movie hide everything behind it on the page. `Transparent` makes the HTML page show through the unfilled portions of the movie and can slow movie performance. Works only in Internet Explorer and only for Windows users.
BASE	Optional. Specifies the base directory or URL used to resolve all relative path statements in the Flash Player movie. This attribute is helpful when your Flash movies are kept in a different directory from your other files.
DEVICEFONT	Optional. If set to `true`, antialiased (smooth-edged) system fonts are substituted for device fonts—such as `_sans` and `_serif`—not installed on the user's system. If `false` (the default), device fonts are not antialiased. Currently, this setting has an effect only when displayed on Windows machines.
SWLIVECONNECT	Optional. Values are `true` and `false`. Whether Java should be started when the page first appears; you should set this to `true` if you will use scripting to control the movie. See the Flash documentation for details.

Note

tag)>A few additional properties are available, including `SCALE`, `ALIGN`, and `SALIGN`, which have to do with how the movie is scaled in certain—generally unusual—situations. See the Flash documentation for details.

Listing 26.10 shows how to use the <OBJECT>, <PARAM>, and <EMBED> tags to include a Flash movie file called owsAnimation.swf. The results are shown in Figure 26.6. For your convenience, the owsAnimation.swf file has been included on this book's CD-ROM to enable you to easily test this template.

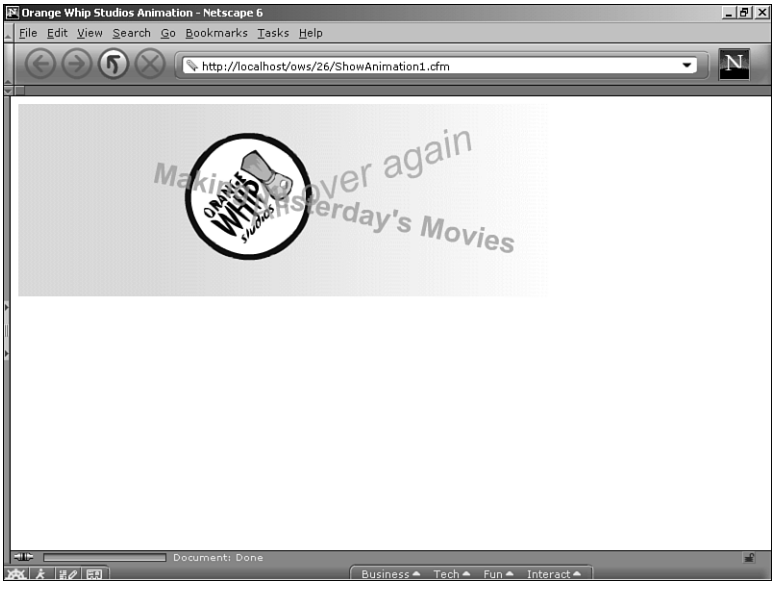

Figure 26.6
Including a Flash movie in your application pages is easy.

LISTING 26.10 ShowAnimation1.cfm—INCLUDING A FLASH MOVIE IN A COLDFUSION TEMPLATE

```
<HTML>
<HEAD>
<TITLE>Orange Whip Studios Animation</TITLE>
</HEAD>
<BODY>

<!--- Include Flash movie --->
<!--- The <OBJECT> and <PARAM> tags are for Internet Explorer --->
<OBJECT
  CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  CODEBASE="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab
➥#version=5,0,0,0"
  WIDTH="550"
  HEIGHT="200">
  <PARAM NAME="movie" VALUE="owsAnimation.swf">
  <PARAM NAME="loop" VALUE="false">
  <PARAM NAME="quality" VALUE="high">
  <PARAM NAME="bgcolor" VALUE="#FFFFFF">
  <PARAM NAME="wmode" VALUE="opaque">

  <!--- The <EMBED> tag is for Netscape browsers --->
  <EMBED
```

LISTING 26.10 CONTINUED

```
    SRC="owsAnimation.swf"
    LOOP="false"
    QUALITY="high"
    BGCOLOR="#FFFFFF"
    WIDTH="550"
    HEIGHT="200"
    TYPE="application/x-shockwave-flash"
    PLUGINSPAGE="http://www.macromedia.com/shockwave/download/index.cgi?
➥P1_Prod_Version=ShockwaveFlash"></EMBED>

</OBJECT>

</BODY>
</HTML>
```

Tip

If you have Macromedia Flash, you can use its Publish command to produce this code automatically. See the Flash documentation for details.

As you can see, the values for the `WIDTH`, `HEIGHT`, `LOOP`, `QUALITY`, and other properties from Table 26.9 are supplied twice. First, the property is provided as an attribute of the `<OBJECT>` tag (for `WIDTH` and `HEIGHT`) or as a `<PARAM>` tag (for the other properties). Then, the property is provided again as an attribute of the `<EMBED>` tag.

Note

You always should provide `CLASSID` and `CODEBASE` attributes for the `<OBJECT>` tag and `TYPE` and `PLUGINSPAGE` attributes for the `<EMBED>` tag, as shown in this section. The values of these attributes always are the same, regardless of the Flash movie you are displaying.

INCLUDING FLASH MOVIES USING A CUSTOM TAG

A custom tag called `<CF_EmbedFlashMovie>` has been included on the CD-ROM for this chapter. The custom tag makes including a Flash movie in one of your ColdFusion templates much easier, outputting all the necessary `<OBJECT>`, `<PARAM>`, and `<EMBED>` code for you.

The attributes supported by the custom tag are shown in Table 26.10. Don't worry about the large number of attributes in this table. In many cases, you will need to use only the `MovieFile`, `Width`, and `Height` attributes.

Note

Because the job of the `<CF_EmbedFlashMovie>` custom tag is to output the appropriate `<OBJECT>`, `<PARAM>`, and `<EMBED>` code for you, most of these attributes correspond closely to the properties listed in Table 26.9.

TABLE 26.10 ATTRIBUTES SUPPORTED BY THE <CF_EmbedFlashMovie> CUSTOM TAG

Attribute	Description
MovieFile	The relative URL path to the Flash Player (.swf) file you want to display.
Width	The width at which you want to show the movie, in pixels (such as WIDTH="50") or as a percentage (such as WIDTH="100%").
Height	The height at which you want to show the movie, in pixels or as a percentage.
Play	Optional. Yes or No. Whether the movie starts automatically when the page appears. Default is Yes.
Loop	Optional. Yes or No. Whether the movie loops repeatedly or just plays once. Default is Yes.
Menu	Optional. Yes or No. Whether the default pop-up menu is available if the user right-clicks the movie. Default is No.
DeviceFont	Optional. Yes or No. Whether device fonts are shown using anti-aliased type on Windows systems. Default is No.
Quality	Can be set to low, high, autolow, autohigh, or best. The default value of high is usually the most appropriate. See the Flash documentation for details.
BGColor	Specifies an optional background color for the movie, which overrides the background color applied when the movie was created.
MovieName	Optional. A name for the movie, if you want to be able to control the movie via script. The value you provide is set as the ID attribute of the generated <OBJECT> tag and the NAME attribute of the generated <EMBED> tag.
Style	Optional CSS-style information.
IndentCode	Optional. Yes or No. Whether the <OBJECT> and <EMBED> code generated by the custom tag should be indented nicely. Default is No.
IncludeObject	Optional. Yes or No. Whether the custom tag should generate <OBJECT> and <PARAM> tags, which enables the movie to be seen by Internet Explorer users. Default is Yes.
IncludeEmbed	Optional. Yes or No. Whether the custom tag should generate <EMBED> code, which enables the movie to be seen by Netscape users. Default is Yes.
EnableFSCommand	Optional. Yes or No. Whether the custom tag should generate the <SCRIPT> blocks necessary for the Flash movie to be capable of carrying out fscommand operations via ActionScript in the movie file itself. If Yes, you can include actual fscommand code between opening and closing <CF_EmbedFlashMovie> tags. This attribute gets set to Yes automatically if you use opening and closing <CF_EmbedFlashMovie> tags. See the section "Interacting with the Underlying Web Page via fscommand," later in this chapter.
swLiveConnect	Optional. Yes or No. Whether Java should be started when the page first appears. Defaults to the value of EnableFSCommand. See the Flash documentation for details.
MovieVar	Relevant only if EnableFSCommand="Yes". Optional string that can be used to refer to the player object from script in an fscommand operation. Default is player.

Note

The custom tag also supports the same `Quality`, `BGColor`, `Scale`, `Align`, `SAlign`, `Base`, and `WMode` attributes that are supported by the Flash player. Refer to Table 26.9 for details.

Listing 26.11 shows how to use the `<CF_EmbedFlashMovie>` custom tag. As you can see, this code is much more straightforward than the previous version of this template (refer to Listing 26.9).

Note

For this example to work, the `EmbedFlashMovie.cfm` file (on the CD-ROM for this chapter) must be placed in the special `CustomTags` folder or in the same folder as Listing 26.11 itself. See Chapter 22 for more information about the special `CustomTags` folder and CFML custom tags in general.

LISTING 26.11 `ShowAnimation2.cfm`—USING THE `<CF_EmbedFlashMovie>` CUSTOM TAG

```
<HTML>
<HEAD>
<TITLE>Orange Whip Studios Animation</TITLE>
</HEAD>
<BODY>

<!--- Include Flash movie, via Custom Tag --->
<!--- Automatically generates all needed <OBJECT>, <PARAM> and <EMBED> tags --->
<CF_EmbedFlashMovie
  MovieFile="owsAnimation.swf"
  Width="550"
  Height="200">

</BODY>
</HTML>
```

If you visit this template with your browser, you will find that the results are the same as those shown previously in Figure 26.6. However, if you use your browser's View Source command, you will find that the HTML code that was actually sent to the browser is similar to the code shown in Listing 26.9. This is yet another example of how ColdFusion's excellent custom tag feature can simplify your life as a coder.

Tip

`<CF_EmbedFlashMovie>` is written in CFML, so you can feel free to adapt it to suit your own needs. See Chapter 22 for details about how to write CFML custom tags.

LOADING VARIABLES FROM COLDFUSION INTO A FLASH MOVIE

Note

This section assumes that you know a bit about ActionScript, which is Flash's internal scripting language for creating interactive movies. In particular, you should be familiar with the `loadVariables` action and with the Flash concepts of timelines and events. You might need to consult the Flash ActionScript Reference (part of the online Help in the Macromedia Flash IDE) to follow along.

If you are familiar with JavaScript, you will find ActionScript's semantics refreshingly similar.

You can integrate a Flash Movie with ColdFusion in a number of ways. Perhaps the easiest way is with the `loadVariables` action provided by ActionScript, which enables a Flash movie to grab values from a URL. Rather than being a conventional Web page, the URL should return just a single line of code, which specifies the names and values of variables that should be set in the Flash movie (see the Flash ActionScript Reference for details).

Therefore, if you create a ColdFusion template called `ExposeFilmInfo.cfm`, and the template returns a single line of code in the proper format, the Flash movie will be capable of grabbing values from ColdFusion in real time just by calling `loadVariables`. The `loadVariables` calls might be in response to user actions, such as mouse clicks, or attached to particular frames in the movie's timelines.

The format required by `loadVariables` is a simple one. It's the same format you use to pass URL parameters to your ColdFusion templates. Each variable is passed with an = sign between the variable's name and its value. If a variable's value contains spaces or other funny characters, they must be encoded using the same encoding scheme normally used in URLs.

So, if your `ExposeFilmInfo.cfm` template returns output that looks like this (without any HTML tags, whitespace, or other characters)

```
NAME=Nate%20Weiss&AGE=32
```

then a Flash movie can use a command like this, which sets variables named `NAME` and `AGE` in the current movie clip:

```
// Load variables from ColdFusion
loadVariables ("ExposeFilmInfo.cfm", this);
```

The Flash movie could then refer to `NAME` and `AGE` as normal `ActionScript` variables. The value of `NAME` will be `Nate Weiss`, and the value of `AGE` will be `32`.

EXPOSING COLDFUSION VARIABLES TO FLASH

Listing 26.12 creates a convenient CFML custom tag called *<CF_ExposeDataToFlash>*, which converts the contents of a ColdFusion structure or query to the URL-encoded format needed by Flash's `loadVariables` action. The tag takes one attribute, `Data`, which should be a structure or a query object.

Note

Don't worry too much about understanding the code in this listing right now. You will be able to use this custom tag without knowing exactly what it is doing internally. You can always study it more closely later.

LISTING 26.12 ExposeDataToFlash.cfm—**CREATING THE** <CF_ExposeDataToFlash> **CUSTOM TAG**

```
<!--- Tag attributes --->
<CFPARAM NAME="ATTRIBUTES.Data" TYPE="any">

<!--- If the passed-in value is a structure --->
<CFIF IsStruct(ATTRIBUTES.Data)>
  <!--- Start with an empty string --->
  <CFSET FinalString = "">

  <!--- For each value in the Data, add key/value pair to FinalString --->
  <CFLOOP COLLECTION="#ATTRIBUTES.Data#" ITEM="ThisKey">
    <!--- Get the value of this key/value pair --->
    <CFSET ThisValue = ATTRIBUTES.Data[ThisKey]>
    <!--- Make sure it's possible to express this value as a string --->
    <CFIF IsSimpleValue(ThisValue)>
      <!--- Add key/value pair to FinalString, with the value encoded --->
      <CFSET ThisPair = "#ThisKey#=#URLEncodedFormat(ThisValue)#">
      <CFSET FinalString = ListAppend(FinalString, ThisPair, "&")>
    </CFIF>
  </CFLOOP>


<!--- If the passed-in value is a query --->
<CFELSEIF IsQuery(ATTRIBUTES.Data)>
  <!--- Start by providing the record count --->
  <CFSET FinalString = "RecordCount=#ATTRIBUTES.Data.RecordCount#">

  <!--- For each row in the query --->
  <CFLOOP QUERY="ATTRIBUTES.Data">
    <!--- Loop through the column names --->
    <CFLOOP LIST="#ATTRIBUTES.Data.ColumnList#" INDEX="ThisCol">
      <!--- Add key/value pair to FinalString, with the value encoded --->
      <!--- Resulting variable names will be in the form COLUMNNAME_ROW --->
      <CFSET ThisValue = Attributes.Data[ThisCol][CurrentRow]>
      <CFSET ThisPair = "#ThisCol#_#CurrentRow#=#URLEncodedFormat(ThisValue)#">
      <CFSET FinalString = ListAppend(FinalString, ThisPair, "&")>
    </CFLOOP>
  </CFLOOP>

<!--- Show an error message if value was not a structure or a query --->
<CFELSE>
  <CFTHROW
    MESSAGE="Error Encountered by &lt;CF_ExposeDataToFlash&gt;"
    DETAIL="The value of the Data attribute must be a structure or a query.
            Don't forget to use pound signs around the query or structure name.">
</CFIF>

<!--- Output final string, using <CFCONTENT> to discard any prior output --->
<CFCONTENT TYPE="text/html" RESET="Yes"><CFOUTPUT>&#FinalString#</CFOUTPUT>
```

If the passed-in `Data` attribute is a structure, the tag creates a variable called `FinalString` and then loops through the key in the structure, adding the names and values for each key in the proper format. The `URLEncodedFormat()` function is used to encode any special characters in the values. At the bottom of the template, the `FinalString` variable is output, preceded with a `<CFCONTENT>` tag with `RESET="Yes"` to discard any prior output. This ensures that the value of `FinalString` appears on the very first line of the generated page. (For more information about `<CFCONTENT>`, see Chapter 34.)

If `Data` is a query object, the tag populates the `FinalString` variable a bit differently. It includes a name/value pair for a variable called `RecordCount`, the value of which is the number of rows in the query. Then, a separate name/value pair is included for each column and row in the query results, using the column name and row number as the name of the pair. Because of the way ColdFusion's automatic `ColumnList` property works, the column names always are represented in uppercase. For instance, if the query contains two columns named `FilmID` and `MovieTitle`, and the query returns three rows, `FinalString` includes pairs for `RecordCount`, `FILMID_1`, `FILMID_2`, `FILMID_3`, `MOVIETITLE_1`, `MOVIETITLE_2`, and `MOVIETITLE_3`.

This custom tag can be tested with the code in Listing 26.13. It creates a structure called `s` with two values, `Name` and `Age`. The structure is then passed to the `<CF_ExposeDataToFlash>` custom tag.

LISTING 26.13 `ExposeDataTest.cfm`—TESTING THE `<CF_ExposeDataToFlash>` CUSTOM TAG

```
<!--- Create structure --->
<CFSET s = StructNew()>
<CFSET s.Age = 32>
<CFSET s.Name = "NateWeiss">

<!--- Output data in URL-like format expected by Flash Player --->
<CF_ExposeDataToFlash
  Data="#s#">
```

If you visit this template with your browser, it outputs a single line, which looks like this:

```
AGE=32&NAME=Nate%20Weiss
```

Note

Because of the way ColdFusion stores structures internally, the name part of each name/value pair is uppercase.

Now all you need is a Flash movie that uses the `loadVariables` action to fetch the values from the `ExposeDataTest.cfm` template shown in Listing 26.13.

Note

You need Macromedia Flash 5 to follow along. A trial version is available at `http://www.macromedia.com`.

To create a Flash movie that fetches data from `ExposeDataTest.cfm`, follow these steps:

1. In Macromedia Flash, create a new movie and save it as `LoadVariablesTest.fla` in the same folder you're using for this chapter's ColdFusion templates.
2. In the Timeline, select the first frame of the movie; then open the Frame Actions panel by selecting Window, Actions from the main menu.
3. Add a `loadVariables` action. For the URL, provide `ExposeDataTest.cfm`. Leave the Location at Level 0 and the Variables field at Don't Send.
4. Place a text object on the stage. Using the Text Options panel (which you can reveal using Window, Panels if it is not showing already), change the type from Static Text to Dynamic Text. For the Variable field, type **`Name`**.
5. Repeat step 4, this time providing `Age` for the variable field.
6. If you want, add labels for the two text fields you just added, or whatever formatting you feel is appropriate, as shown in Figure 26.7.
7. Save the movie, and then publish it by pressing Shift+F12. Verify that a file called `LoadVariablesTest.swf` was created in the folder you are using for this chapter's listings.

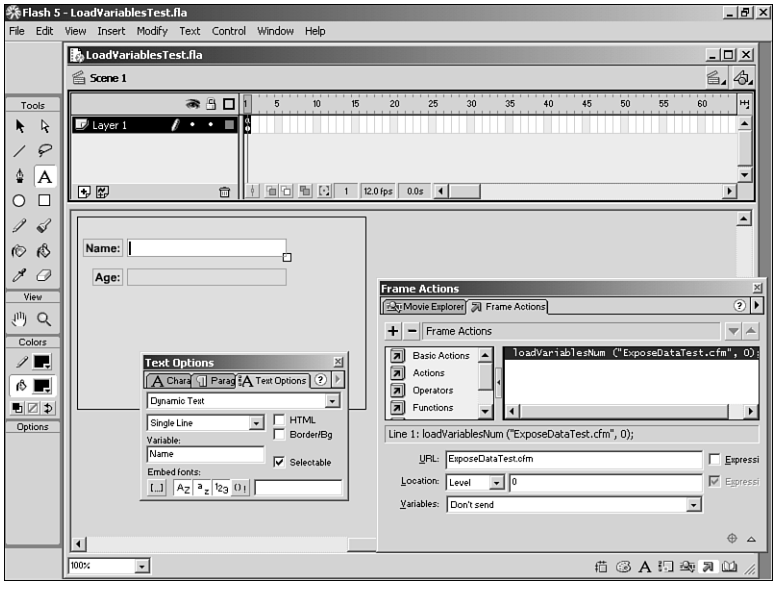

Figure 26.7
Text fields can be used to reflect the values of variables retrieved with the `loadVariables` action.

Now you can test the movie with the code shown in Listing 26.14. Note that this is essentially the same code used in Listing 26.11.

LISTING 26.14 `LoadVariablesTest.cfm`—DISPLAYING THE `LoadVariablesTest.swf` FLASH MOVIE

```
<HTML>
<HEAD>
<TITLE>loadVariables Test</TITLE>
</HEAD>
<BODY>

<!--- Include Flash movie, via Custom Tag --->
<!--- Automatically generates all needed <OBJECT>, <PARAM> and <EMBED> tags --->
<CF_EmbedFlashMovie
  MovieFile="LoadVariablesTest.swf"
  Width="300"
  Height="200">

</BODY>
</HTML>
```

When you visit this template with your browser, the Flash movie you created (`LoadVariablesTest.swf`) should display with the appropriate name and age values, as shown in Figure 26.8. If you go back and edit Listing 26.13 (so that the name and age values are different) and then reload this template, you should see the values change in the Flash movie.

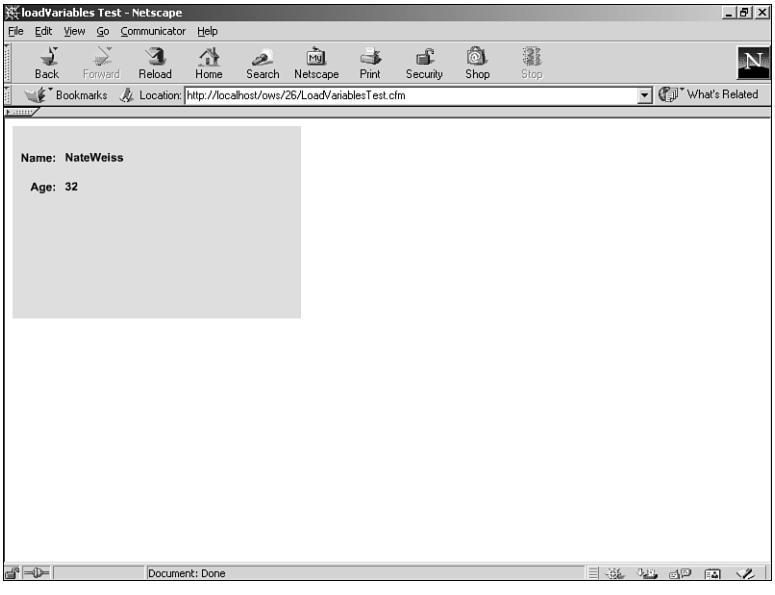

Figure 26.8
Flash movies can fetch variables from ColdFusion templates.

Note

The Flash movie you created in this section is very simple and just fetches the values once when the movie first appears. Of course, you could choose to use the `loadVariables` action at any point in your movie or in response to some type of user interaction, such as a button press.

WORKING WITH A QUERY OBJECT

The `FilmChooser.fla` file included on the CD-ROM for this chapter creates a Flash Player file called `FilmChooser.swf` that fetches query data from a ColdFusion template. The basic technique is the same as demonstrated in the previous section, except this time Flash's `loadVariables` action is used to fetch data from a template that supplies a query object to the `<CF_ExposeDataToFlash>` custom tag (instead of a structure).

Listing 26.15 provides the code for a ColdFusion template called `ExposeFilmInfo.cfm`, which is used by the `FilmChooser.swf` movie via the `loadVariables` action. It simply runs a query named `GetFilms` and then passes the query object to the `<CF_ExposeDataToFlash>` custom tag. If you visit this template with your browser, you will find that a long string of data is returned, including the `FilmID` and `MovieTitle` for each film in the `Films` table.

LISTING 26.15 `ExposeFilmInfo.cfm`—SUPPLYING QUERY INFORMATION FOR THE FLASH `loadVariables` ACTION

```
<!--- Select film titles from database --->
<CFQUERY NAME="GetFilms" DATASOURCE="ows"
  CACHEDWITHIN="#CreateTimeSpan(0,0,5,0)#">
  SELECT FilmID, MovieTitle
  FROM Films
  ORDER BY MovieTitle DESC
</CFQUERY>

<!--- Output query data in URL-like format expected by Flash Player --->
<CF_ExposeDataToFlash
  Data="#GetFilms#">
```

Unfortunately, it isn't possible for us to explain all the ActionScript code used in the `FilmChooser.swf` Flash movie (ActionScript is well beyond the scope of this book). That said, you should be able to understand most of what is going on just by exploring the `FilmChooser.fla` file (perhaps consulting the Flash documentation as you go). The most important code, as far as integration with ColdFusion is concerned, is attached to the first two frames of the movie's main timeline.

In the first frame of the timeline, the `loadVariables` action is used to fetch the variables from the `ExposeFilmInfo.cfm` template (Listing 26.15), like so:

```
// Load variables from ColdFusion
loadVariables ("ExposeFilmInfo.cfm", this);
```

The second frame of the timeline uses the ActionScript code shown in Listing 26.16 to loop through the variables produced by Listing 26.15. Because it is in the second frame of the timeline, this code executes after the `loadVariables` action.

LISTING 26.16 ACTIONSCRIPT CODE USED IN THE SECOND FRAME OF `FilmChooser.swf`

```
// If loadVariables call in previous frame fetched a RecordCount
if (RecordCount > 0) {

  // Loop through records
  for (i=1; i<RecordCount; i++) {
    // Get this FilmID and MovieTitle
    ThisFilmID = eval("FILMID_" add i);
    ThisMovieTitle = eval("MOVIETITLE_" add i);

    // Create new copy of FilmDisplay movie clip
    ClipName = "Film" add ThisFilmID;
    duplicateMovieClip ("FilmDisplay", ClipName, i);

    // Set the title in the new movie clip
    set (ClipName add ":FilmID", ThisFilmID);
    set (ClipName add ":MovieTitle", ThisMovieTitle);
  }

  // Make original FilmDisplay movie clip invisible
  setProperty ("FilmDisplay", _visible, 0);

  // Stop animation in main timeline
  // (the FilmDisplay clips will move around on their own)
  stop ();
}
```

The basic idea in this template is to use the `RecordCount` variable (which is one of the variables automatically exposed by the `ExposeFilmInfo.cfm` template from Listing 26.15) to loop through the query data fetched by the first frame's `loadVariables` action. A simple `for` loop is used to do the looping; for each iteration through the loop, the value of the `i` variable holds the current row number.

As explained just before Listing 26.13, the values from the first row of the query are available as `FILMID_1` and `MOVIETITLE_1`. The first two lines inside the `for` loop use the ActionScript `eval` function (which is comparable to the `Evaluate()` function in CFML) to get the value of those two variables and set them as new variables named `ThisFilmID` and `ThisMovieTitle`.

Next, a new movie clip is created using the `duplicateMovieClip` action. The new clip is a copy of the clip named `FilmDisplay` (which is located just off the stage). The `set` statement is then used to set local variables called `FilmID` and `MovieTitle` in the new clip's timeline. Because the clip contains a text field that displays the value of the `MovieTitle` variable, the film's title is now visible to the user. The loop then repeats for all the remaining film records (that is, until `i` reaches the value of `RecordCount`).

After the loop is finished, the `FilmDisplay` movie clip (which has just been copied once for each film) is made invisible, and then the movie's main timeline is halted with the `stop` action. The end result is that now a separate movie clip is on the stage for each film. Each of the clips—because of code in the clips symbol itself, which we don't have space to explain here—floats around on the stage on its own and responds to the user's mouse movements and clicks.

Note

The `FilmDisplay` movie clip has a bit of ActionScript in its own timeline, which causes it to move around on the stage. Although the code is certainly not sophisticated as far as the larger world of Flash programming is concerned, you are encouraged to take a glance at it if you are at all unfamiliar with how to make things move around in a Flash movie.

All that is left to do now is to display the movie on a Web page, which can be done with the `<CF_EmbedFlashMovie>` custom tag as shown in Listing 26.17. When this template is visited in a Web browser, the Flash movie begins playing automatically.

LISTING 26.17 `ShowFilmChooser1.cfm`—DISPLAYING THE `FilmChooser.swf` MOVIE

```
<HTML>
<HEAD><TITLE>Film Animation</TITLE></HEAD>
<BODY>

  <!--- Show Flash movie --->
  <CF_EmbedFlashMovie
    MovieFile="FilmChooser.swf"
    Width="100%"
    Height="100%">

</BODY>
</HTML>
```

Assuming that Listing 26.15 has been saved in the same folder as this listing, the titles of each movie will creep onto the page, each floating at its own speed. The titles will continue to move around, bouncing from side to side and top to bottom. If you hover your mouse over one of the titles, it moves forward and stops, as shown in Figure 26.9. When you move your mouse away from the title, it begins moving again. This provides the user with a fun, unconventional way to browse through the list of films.

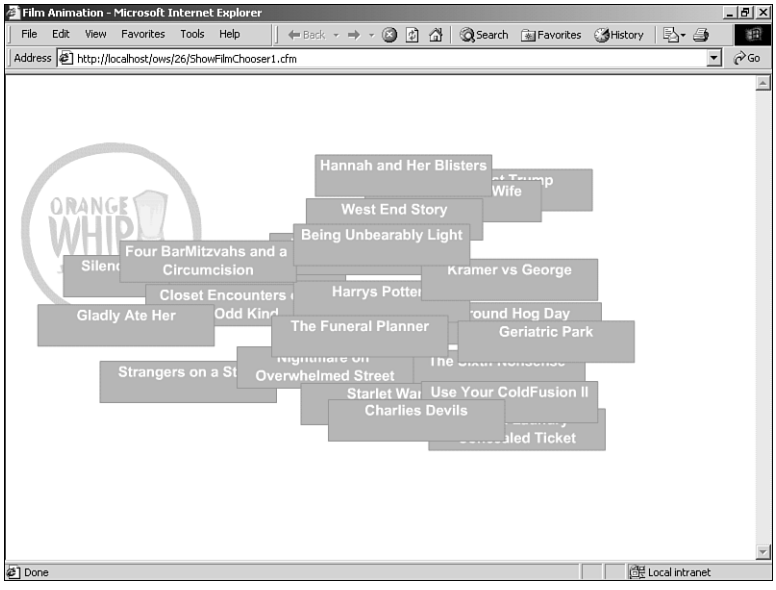

Figure 26.9
Flash movies can fetch query data from ColdFusion templates, via the `loadVariables` action.

INTERACTING WITH THE UNDERLYING WEB PAGE VIA `fscommand`

Another way to get a Flash movie to interact with ColdFusion is via the `fscommand` action provided by ActionScript. The `fscommand` action is a simple mechanism that enables a Flash movie to execute JavaScript code on the Web page on which the movie is being displayed.

If you open the `FilmChooser.fla` file and look at the Invisible Button layer of the `FilmRecord` symbol's timeline, you will see that it includes an invisible button symbol that includes the following lines of ActionScript code:

```
on (release) {
    // Show detail when user clicks on symbol
    fscommand ( "showFilmDetail", FilmID );
}
```

What this means is that the `fscommand` action executes when the user clicks a film's title (refer to Figure 26.9). This causes the Flash movie to attempt to call a special script function on the Web page. The function is passed two parameters, `command` and `args`. In this example, the `command` is `showFilmDetail` and `args` is the `FilmID` of the film the user clicks.

Tip

For more information about `fscommand`, see the Flash documentation.

Normally, you would need to insert a number of special `<SCRIPT>` blocks and other code before you could respond to an `fscommand` action. If you use the `<CF_EmbedFlashMovie>` custom tag to display the movie, however, this is all taken care of for you.

Simply add a closing </CF_EmbedFlashMovie> tag to the template. Now add one or more <CF_EmbedFlashMovieCommand> tags between the opening and closing <CF_EmbedFlashMovie> tags. The <CF_EmbedFlashMovieCommand> takes two arguments, as shown in Table 26.11.

TABLE 26.11 ATTRIBUTES SUPPORTED BY THE <CF_EmbedFlashMovieCommand> CUSTOM TAG

Attribute	Description
Command	The first argument of the fscommand action to which you want to respond. If the movie executes the command, the JavaScript code between the opening and closing <CF_EmbedFlashMovieCommand> tags is executed. Please note that the value of this attribute is case sensitive.
ArgsVariable	A variable name you want to be able to use to refer to the second argument of the fscommand action. You can then use the variable in the JavaScript code between the <CF_EmbedFlashMovieCommand> tags, effectively enabling you to pass values from the Flash movie to your JavaScript code.

Listing 26.18 shows how easy responding to fscommand actions is with the <CF_EmbedFlashMovieCommand> custom tag. Note that the value of the Command attribute matches the first argument of the fscommand action call (refer to the code snippet before Table 26.11).

Note

For this example to work, the EmbedFlashMovieCommand.cfm file (on the CD-ROM for this chapter) must be placed in the special CustomTags folder or in the same folder as Listing 26.11 itself. See Chapter 22 for more information about the special CustomTags folder and CFML custom tags in general.

LISTING 26.18 ShowFilmChooser2.cfm—RESPONDING TO fscommand ACTIONS

```
<HTML>
<HEAD><TITLE>Film Animation</TITLE></HEAD>
<BODY>

  <!--- Show Flash movie --->
  <CF_EmbedFlashMovie
    MovieFile="FilmChooser.swf"
    Width="100%"
    Height="100%"
    IndentCode="Yes">

    <!--- If the movie executes an FSCommand of command "showFilmDetail" --->
    <CF_EmbedFlashMovieCommand
      Command="showFilmDetail"
      ArgsVariable="FilmID">
      var url = "http://"+location.host+"/ows/26/ShowFilm.cfm?FilmID=" + FilmID;
      window.open(url, "wFilm", "width=250,height=200,scrollbars=yes");
    </CF_EmbedFlashMovieCommand>
```

```
    </CF_EmbedFlashMovie>

</BODY>
</HTML>
```

Tip

For more information about the `window.open` and `location.host` code used in this listing, consult a JavaScript reference text or visit the online JavaScript documentation at `http://developer.netscape.com`.

When the user clicks a film title, the `fscommand` action shown previously is executed. This causes the JavaScript code between the `<CF_EmbedFlashMovieCommand>` to execute. The JavaScript code itself is quite simple. It opens a ColdFusion template called `ShowFilm.cfm` in a small pop-up window, passing the `FilmID` of the selected movie as a URL parameter.

The `ShowFilm.cfm` template displays the details for the film the user selected. In other words, the user is able to click the floating film titles to see more information about each film, as shown in Figure 26.10.

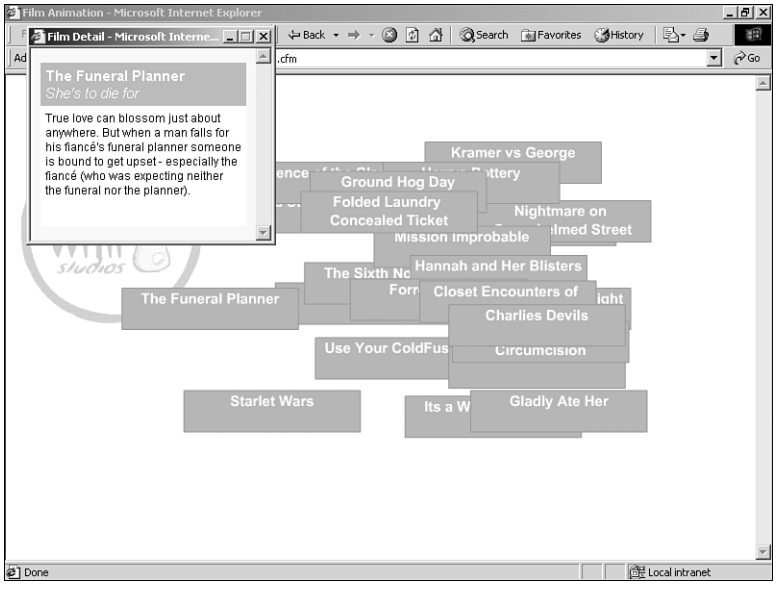

Figure 26.10
The `fscommand` action can be used to pop up browser windows or other JavaScript interactions.

Listing 26.19 provides the code for the `ShowFilm.cfm` template displayed in the pop-up window for each film (refer to Figure 26.10). There is nothing particularly special about this template. It simply queries the database for the details about the specified film and then displays the information in a simple HTML table.

Note

The code in this template uses the Query-of-Queries feature to requery the cached `GetFilms` query for each request, rather than recontacting the database separately for each film. See Chapter 31, "Advanced Querying," for more information.

LISTING 26.19 `ShowFilm.cfm`—DISPLAYING FILM INFORMATION IN A POP-UP WINDOW

```
<!--- FilmID must be passed in URL --->
<CFPARAM NAME="URL.FilmID" TYPE="numeric">

<!--- Retrieve information about films from database --->
<CFQUERY NAME="GetFilms" DATASOURCE="ows"
  CACHEDWITHIN="#CreateTimeSpan(0,0,15,0)#">
  SELECT FilmID, MovieTitle, PitchText, Summary
  FROM Films
</CFQUERY>

<!--- Use Query-of-Queries to grab this film's information --->
<CFQUERY DBTYPE="query" NAME="GetFilm">
  SELECT * FROM GetFilms
  WHERE FilmID = #URL.FilmID#
</CFQUERY>


<HTML>
<HEAD>
  <TITLE>Film Detail</TITLE>

  <!--- Some CSS-based classes for formatting --->
  <STYLE TYPE="text/css">
    .Title {font-family:sans-serif;font-size:15px;background:orange;color:white;}
    .Summary {font-family:sans-serif;font-size:12px;background:white;color:black;
      height:110px}
  </STYLE>
</HEAD>

<BODY
  BGCOLOR="yellow"
  onLoad="focus()">

<!--- Display film information --->
<CFOUTPUT>
  <TABLE WIDTH="100%" HEIGHT="100%" BORDER="0" CELLPADDING="5" CELLSPACING="0">
    <TR HEIGHT="20%">
      <TD CLASS="Title">
        <B>#GetFilm.MovieTitle#</B><BR>
        <I>#GetFilm.PitchText#</I>
      </TD>
    </TR>
    <TR HEIGHT="80%">
      <TD CLASS="Summary" VALIGN="top">
        #GetFilm.Summary#
      </TD>
    </TR>
  </TABLE>
```

```
</CFOUTPUT>

</BODY>
</HTML>
```

Tip

The call to `focus()` in the `onLoad` attribute of the `<BODY>` tag ensures that the pop-up window appears in front of the main browser window. It's a handy line of code to add to any template that will be displayed in a pop-up.

Note

The `fscommand` action does not work with all browsers, especially Netscape 6 and Internet Explorer on the Macintosh. For details and alternatives, see TechNote Article 14159 at `http://www.macromedia.com/support/flash/`.

OTHER TOPICS

Although you have learned a lot in the second half of this chapter, these examples have really only scratched the surface. You can get Flash and ColdFusion to work together in many ways. If this topic interests you, you are encouraged to investigate the following:

- **XML support in Flash 5**—The Flash 5 Player enables you to fetch XML content from URLs, somewhat like the `loadVariables` function. The difference, of course, is that XML content can be far more structured, validated, and so on. For details, see the ActionScript Reference portion of the Flash 5 documentation.
- **WDDX support provided by the Component Kit**—One of the files included in the Macromedia Flash Component Kit for ColdFusion is an ActionScript file called `wddx.as`, which sits on top of the Flash Player's native XML support. You can use this script file to fetch data from a ColdFusion template that uses the `<CFWDDX>` tag to expose structured information. Documentation for the `wddx.as` file is somewhat scant as of this writing, but it is conceptually similar to the `wddx.js` file that ships with ColdFusion, which is documented in the WDDX SDK from `http://www.openwddx.org`.
- **Scripting methods supported by the Flash Player**—You can control Flash movie playback, zooming, panning, and so on with the scripting methods described in Article 04160 at `http://www.macromedia.com/support/flash/`.
- **ColdFusion templates as Macromedia Generator data sources**—ColdFusion template URLs easily can be used as data sources for Macromedia Generator templates. For details, download an evaluation copy of Macromedia Generator from `http://www.macromedia.com`.