CHAPTER 11

Scripting Frames and Multiple Windows

In This Chapter

Relationships among frames in the browser window

How to access objects and values in other frames

How to control navigation of multiple frames

Communication skills between separate windows

One of the cool aspects of JavaScript on the client is it allows user actions in one frame or window to influence what happens in other frames and windows. In this section of the tutorial, you extend your existing knowledge of object references to the realm of multiple frames and windows.

Frames: Parents and Children

You probably noticed that at the top of the Navigator document object hierarchy diagram (refer back to Figure 8-1) the window object has some other object references associated with it. Back in Chapter 8 you learned that `self` is synonymous with `window` when the reference applies to the same window that contains the script's document. In this lesson, you'll learn the roles of the other three object references — `frame`, `top`, and `parent`.

Loading an ordinary HTML document into the browser creates a model in the browser that starts out with one window object and the document it contains (the document likely contains other elements, but I'm not concerned with that stuff yet). The top rungs of the hierarchy model are as simple as can be, as shown in Figure 11-1. This is where references begin with `window` or `self` (or with `document`, since the current window is assumed).

Figure 11-1: Single-frame window and document hierarchy

The instant a framesetting document loads into a browser, the browser starts building a slightly different hierarchy model. The precise structure of that model depends entirely on the structure of the frameset defined in that framesetting document. Consider the following skeletal frameset definition:

```
<HTML>
<FRAMESET COLS="50%,50%">
        <FRAME NAME="leftFrame" SRC="somedoc1.html">
        <FRAME NAME="rightFrame" SRC="somedoc2.html">
</FRAMESET>
</HTML>
```

This HTML splits the browser window into two frames side by side, with a different document loaded into each frame. The model is concerned only with structure — it doesn't care about the relative sizes of the frames or whether they're set up in columns or rows.

Framesets establish relationships among the frames in the collection. Borrowing terminology from the object-oriented programming world, the framesetting document loads into a *parent* window. Each of the frames defined in that parent window document is a *child* frame (although you won't be using the child term in scripting). Figure 11-2 shows the hierarchical model of a two-frame environment. This illustration reveals a lot of subtleties about the relationships among framesets and their frames.

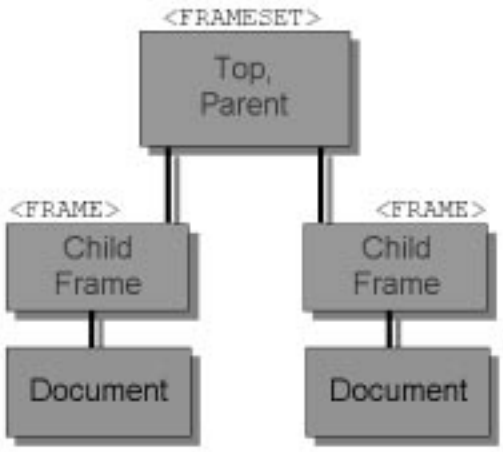

Figure 11-2: Two-frame window and document hierarchy

It is often difficult at first to visualize the frameset as a window object in the hierarchy. After all, with the exception of the URL showing in the Location field, you don't see anything about the frameset in the browser. But that window object exists in the object model. Notice, too, that the framesetting parent window has no document object. This may also seem odd, since the window obviously requires an HTML file containing the specifications for the frameset. But because the HTML of a framesetting file has no <BODY> tag or other document-centric elements, no document object in this portion of the object model is loaded in the browser.

If you were to add a script to the framesetting document that needed to access a property or method of that window object, references would be like any single-frame situation. Think about the point of view of a script located in that window. Its immediate universe is the very same window.

Things get more interesting when you start looking at the child frames. Each of these frames contains a document object whose content you see in the browser window. And the structure is such that each document is entirely independent of the other. It is as if each document lived in its own browser window. Indeed, that's why each child frame is also a window type of object. A frame has all the properties and methods of the window object that occupies the entire browser.

From the point of view of either child window in Figure 11-2, its immediate container is the parent window. When a parent window is at the very top of the hierarchical model loaded in the browser, that window is also referred to as the top object.

References among Family Members

Given the frame structure of Figure 11-2, it's time to look at how a script in any one of those windows would access objects, functions, or variables in the others. An important point to remember about this facility is that if a script has access to an object, function, or global variable in its own window, that same item can be reached by a script from another frame in the hierarchy (provided both documents come from the same Web server).

A script reference may need to take one of three possible routes in the two-generation hierarchy described so far: parent to child; child to parent;child to child.

Each of the paths between these windows requires a different reference style.

Parent-to-child references

Probably the least common direction taken by references is when a script in the parent document needs to access some element of one of its frames. From the point of view of the parent, it contains two or more frames, which means the frames are also stored in the model as an array of frame objects. You can address a frame by array syntax or by the name you assign to it with the `NAME` attribute inside the `<FRAME>` tag. In the following examples of reference syntax, I substitute a placeholder named *`ObjFuncVarName`* for whatever object, function, or global variable you intend to access in the distant window or frame. Remember that each visible frame contains a document object, which is generally the container of elements you will be scripting — be sure references include the document. With that in mind, a reference from a parent to one of its child frames follows either of the following models:

```
[window.]frames[n].ObjFuncVarName
[window.]frameName.ObjFuncVarName
```

Index values for frames are based on the order in which their `<FRAME>` tags appear in the framesetting document. You will make your life easier, however, if you assign recognizable names to each frame and use the frame's name in the reference. Some problems also existed in early scriptable browsers with including

the window reference at the start of all of the references described in this chapter. I recommend omitting `window` from all such references.

Child-to-parent references

It is not uncommon to place scripts in the parent (in the Head portion) that multiple child frames or multiple documents in a frame use as a kind of script library. By loading in the frameset, they load only once while the frameset is visible. If other documents load into the frames over time, they can take advantage of the parent's scripts without having to load their own copies into the browser.

From the child's point of view, the next level up the hierarchy is called the `parent`. Therefore, a reference to items at that level is simply

```
parent.ObjFuncVarName
```

If the item accessed in the parent is a function that returns a value, the returned value transcends the parent-child borders without hesitation.

When the parent window is also at the very top of the object hierarchy currently loaded into the browser, you can optionally refer to it as the top window, as in

```
top.ObjFuncVarName
```

Using the `top` reference can be hazardous if for some reason your Web page gets displayed in some other Web site's frameset. What is your top window is not the master frameset's top window. Therefore, I recommend using the parent reference whenever possible.

Child-to-child references

The browser needs a bit more assistance when it comes to getting one child window to communicate with one of its siblings. One of the properties of any window or frame is its `parent` (if a parent exists). A reference must use this property to work its way out of the current frame to a point that both child frames have in common — the parent in this case. Once the reference is at the parent level, the rest of the reference can carry on as if starting at the parent. Thus, from one child to one of its siblings, you can use either of the following reference formats:

```
parent.frames[n].ObjFuncVarName
parent.frameName.ObjFuncVarName
```

A reference from the other sibling back to the first would look the same, but the `frames[]` array index or *`frameName`* part of the reference would be different. Of course, much more complex frame hierarchies are possible in HTML. An example of a three-generation frameset is shown in Chapter 14. Even so, the document object model and referencing scheme provides a solution for the most deeply nested and gnarled frame arrangement you can think of — following the same precepts you just learned.

Frame Scripting Tips

One of the first mistakes that frame scripting newcomers make is writing immediate script statements that call upon other frames while the pages are loading. The problem here is there is no guaranteed document loading sequence. All you know for sure is that the parent document begins loading first. Regardless of the order of <FRAME> tags, child frames can begin loading at any time. Moreover, a frame's loading time depends on other elements in the document, such as images or Java applets.

Fortunately, you can use a certain technique to initiate a script once all of the documents in the frameset have completely loaded. Just as the onLoad= event handler for a document fires when that document has fully loaded, a parent's onLoad= event handler fires after the onLoad= event handlers in its child frames have fired. Therefore, you can specify an onLoad= event handler in the <FRAMESET> tag. That handler might invoke a function in the framesetting document that then has the freedom to tap the objects, functions, or variables of all frames throughout the object hierarchy.

Controlling Multiple Frames — Navigation Bars

If you are enamored of frames as a way to help organize a complex Web page, you may find yourself wanting to control the navigation of one or more frames from a static navigation panel. I demonstrate here scripting concepts for such control using an application called Decision Helper (which can be found in the Bonus Applications Chapters folder on the CD-ROM). The application, consists of four frames (see Figure 11-3). The top-left frame is one image that has four graphical buttons in it. The goal is to turn that image into a client-side image map, and script it so the pages change in the two right-hand frames. In the upper-right frame, the script will load an entirely different document along the sequence of five different documents that go in there; in the lower-right frame, the script will navigate to one of five anchors to display the segment of instructions that applies to the document loaded in the upper-right frame.

Listing 11-1 shows a slightly modified version of the actual file for the Decision Helper application. The listing contains a couple of new objects and concepts not yet covered in this tutorial. But as you will see, they are extensions to what you already know about JavaScript and objects. To help simplify the discussion here, I have removed the scripting and HTML for the top and bottom button of the area map. Only the two navigation arrows are covered here.

Look first at the HTML section for the Body portion. Almost everything there is standard stuff for defining client-side image maps. The coordinates form rectangles around each of the arrows in the larger image. The HREF attributes for the two areas point to JavaScript functions defined in the Head portion of the document.

In the frameset that defines the Decision Helper application, names are assigned to each frame. The upper-right frame is called entryForms; the lower-left frame is called instructions.

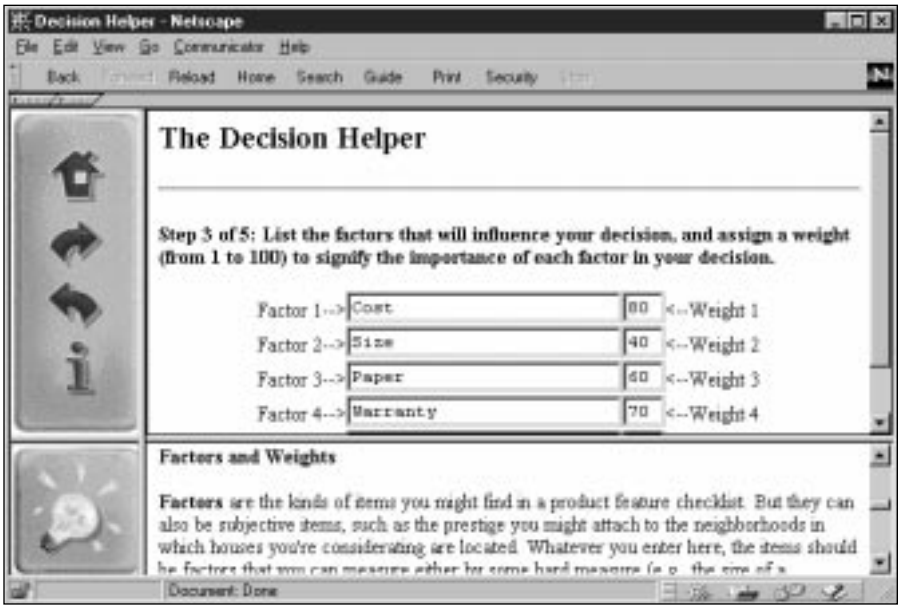

Figure 11-3: The Decision Helper screen

Knowing that navigation from page to page in the upper-right frame would require knowledge of which page is currently loaded there, I built some other scripting into both the parent document and each of the documents that loads into that frame. In the parent document is defined a global variable called `currTitle`. Its value is an integer indicating which page of the sequence (1 through 5) is currently loaded. An `onLoad=` event handler in each of the five documents (named dh1.htm, dh2.htm, dh3.htm, dh4.htm, dh5.htm) assigns its page number to that parent global variable. This arrangement allows that value to be shared easily with all frames in the frameset.

When a user clicks on the right-facing arrow to move to the next page, the `goNext()` function is called. The first statement gets the `currTitle` value from the parent window, and assigns it to a local variable, `currOffset`. An `if...else` construction tests whether the current page number is less than five. If so, the add-by-value operator adds one to the local variable so I can use that value in the next two statements.

In those next two statements, I adjust the content of the two right frames. Using the parent reference to gain access to both frames, I set the location object of the top-right frame to the name of the file next in line (by concatenating the number with the surrounding parts of the filename). The second statement sets the `location.hash` property (a property that controls the anchor being navigated to) to the corresponding anchor in the instructions frame (anchor names `help1`, `help2`, `help3`, `help4`, and `help5`).

A click of the left-facing arrow reverses the process, subtracting 1 from the current page number (using the subtract-by-value operator) and changing the same frames accordingly.

Listing 11-1: **A Graphical Navigation Bar**

```
<HTML>
<HEAD>
<TITLE>Navigation Bar</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!-- start
function goNext() {
        var currOffset = parent.currTitle
        if (currOffset <5) {
           currOffset += 1
           parent.entryForms.location = "dh" + currOffset + ".htm"
           parent.instructions.location.hash = "help" + currOffset
        } else {
           alert("This is the last form.")
        }
}
function goPrev() {
        var currOffset = parseInt(parent.currTitle)
        if (currOffset > 1) {
           currOffset -= 1
           parent.entryForms.location = "dh" + currOffset + ".htm"
           parent.instructions.location.hash = "help" + currOffset
        } else {
           alert("This is the first form.")
        }
}
// end -->
</SCRIPT>
</HEAD>
<BODY bgColor="white">
<MAP NAME="navigation">
<AREA SHAPE="RECT" COORDS="25,80,66,116" HREF="javascript:goNext()">
<AREA SHAPE="RECT" COORDS="24,125,67,161" HREF="javascript:goPrev()">
</MAP>
<IMG SRC="dhNav.gif" HEIGHT=240 WIDTH=96 BORDER=0 USEMAP="#navigation">
</BODY>
</HTML>
```

The example shown in Listing 11-1 is one of many ways to script a navigation frame in JavaScript. Whatever methodology you use, there will be much interaction among the frames in the frameset.

More about Window References

Back in Chapter 8, you saw how to create a new window and communicate with it by way of the window object reference returned from the `window.open()` method. In this section, I introduce you to how one of those subwindows can

communicate with objects, functions, and variables back in the window or frame that created the subwindow.

In scriptable browsers except for Navigator 2, every window has a property called `opener`. The property contains a reference to the window or frame that held the script whose `window.open()` statement generated the subwindow. For the main browser window and frames therein, this value is `null`. Because the `opener` property is a valid window reference, you can use it to begin the reference to items back in the original window, just like a script in a child frame would use `parent` to access items in the parent document. The parent-child terminology doesn't apply to subwindows, however.

Listings 11-2 and 11-3 contain documents that work together in separate windows. Listing 11-2 displays a button that opens a smaller window and loads Listing 11-3 into it. The main window document also contains a text field that gets filled in when you enter text into a corresponding field in the subwindow.

In the main window document, the `newWindow()` function generates the new window. Because no other statements in the document require the reference to the new window just opened, the statement does not assign its returned value to any variable. This is an acceptable practice in JavaScript if you don't need the returned value of a function or method.

Listing 11-2: **A Main Window Document**

```
<HTML>
<HEAD>
<TITLE>Main Document</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function newWindow() {
        window.open("subwind.htm","sub","HEIGHT=200,WIDTH=200")
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button" VALUE="New Window" onClick="newWindow()">
<BR>
Text incoming from subwindow:
<INPUT TYPE="Text" NAME="entry">
<FORM>
</BODY>
</HTML>
```

All of the action in the subwindow document comes in the `onChange=` event handler of the text field. It assigns the subwindow field's own value to the value of the field in the opener window's document. Remember that the contents of each window and frame belong to a document. So even after your reference targets a specific window or frame, the reference must continue helping the browser find the ultimate destination, which is generally some element of the document.

Listing 11-3: **A Subwindow Document**

```
<HTML>
<HEAD>
<TITLE>A SubDocument</TITLE>
</HEAD>
<BODY>
<FORM onSubmit="return false">
Enter text to be copied to the main window:
<INPUT TYPE="text" onChange="opener.document.forms[0].entry.value =
this.value">
</FORM>
</HTML>
```

Just one more lesson to go before I let you explore all the details elsewhere in the book. I'll use the final class to show you some fun things you can do with your Web pages, like changing images when the user rolls the mouse atop a picture.

Exercises

Before answering the first three questions, study the structure of the following frameset for a Web site that lists college courses:

```
<FRAMESET ROWS="85%,15%">
        <FRAMESET COLS="20%,80%">
            <FRAME NAME="mechanics" SRC="history101M..html">
            <FRAME NAME="description" SRC="history101D.html">
        </FRAMESET>
        <FRAMESET COLS="100%">
            <FRAME NAME="navigation" SRC="navigator.html">
        </FRAMESET>
</FRAMESET>
</HTML>
```

1. Whenever a document loads into the description frame, it has an `onLoad=` event handler that stores a course identifier in the framesetting document's global variable called `currCourse`. Write the `onLoad=` event handler that sets this value to "history101".
2. Draw a block diagram that describes the hierarchy of the windows and frames represented in the frameset definition.
3. Write the JavaScript statements located in the navigation frame that load the file "french201M.html" into the mechanics frame and the file "french201D.html" into the description frame.
4. While a frameset is still loading, a JavaScript error message suddenly appears saying that "window.document.navigation.form.selector is undefined." What do you think is happening in the application's scripts, and how can the problem be solved?

5. A script in a child frame of the main window uses `window.open()` to generate a second window. How would a script in the second window access the location object (URL) of the parent window in the main browser window?

✦ ✦ ✦