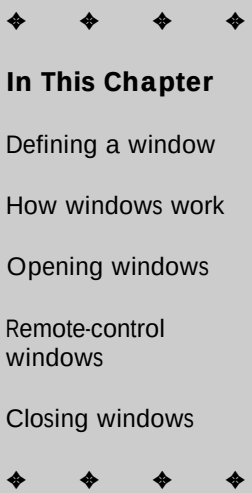
CHAPTER 50

Doing Windows with JavaScript



In This Chapter

Defining a window

How windows work

Opening windows

Remote-control windows

Closing windows

With the predominance of Microsoft products in the marketplace, the word *Windows* has become synonymous with *Microsoft Windows Operating System*. But here in the world of the Web, it's not the only definition. In this chapter, we talk about browser windows. Everything that happens on a Web page happens in context to a browser window.

This chapter shows you how to control the activity within a window and then create and control new windows from there. And you'll see a powerful feature — windows that can talk to each other.

What's a Window?

Browsers use windows to display content. Some pages tell the browser to create new windows for additional content. You've probably seen this in e-commerce sites that ask you to fill out surveys about service, in pop-up advertising on some search engines, and in sites that open a new window when you click on links outside their domain (see Figure 50-1).

Figure 50-1: Clicking on a "related site" link on **www.cnn.com** opens a new browser window.

Everything that happens on a Web page — dynamic or not — takes place in a window. Using JavaScript and DHTML, you can create your own new windows in any size you want, with any control elements you want, and with any content you want. You can also have these windows communicate with one another.

Window Workings

The browser window is referred to in JavaScript with the `window` object. This refers to the current window with the active page. You can use this object to change the contents of the window by loading a new page, utilizing the `location` property.

```
<SCRIPT language="javascript">
window.location = "http://www.idgbooks.com";
</SCRIPT>
```

When the script runs, the contents of the window change from whatever they are currently to the location specified in the string assigned to `window.location` — in this case, the IDG Books Worldwide site.

Of course, at this point the script isn't very useful. It happens automatically before the user has a chance to do anything about it. We could add it to an `if` statement to load the IDG page if another condition is met.

```
<SCRIPT language="javascript">
if ( idgInfo ) {
window.location = "http://www.idgbooks.com";
}
</SCRIPT>
```

If the value of `idgInfo` is non-zero, then the current window loads `http://www.idgbooks.com`, just as if the user had clicked on a hyperlink with that URL as its *href*, or if they had typed the URL into the address field of the browser directly.

With this key concept under your belt, let's create a new window from the existing one.

Opening Windows

JavaScript can create and control more than one window as easily as it controls one. How does this happen? Take a look at the following code snippet:

```
<SCRIPT language="JavaScript">
var winNew;
function openNew() {
winNew = window.open('http://www.idgbooks.com', 'newWindow')
   }
</SCRIPT>
<!-- other HTML here -->
<A href= "javascript:openNew()">Open a Window</A>
```

Clicking on the hyperlink opens a new window with a new file (see Figure 50-2).

As you can see, this window looks different from a "normal" browser window. This happens with the `window.open` method. Table 50-1 lists one key property of this method — file name. This identifies the name of a file or URL to open into the new window. Without this property, the browser would open a blank window without content.

Figure 50-2: A new window is opened when a JavaScript function runs to create it.

Table 50-1
Window.open() Properties

Property	*Definition*
File name	URL of a file to be displayed in the window
Target name	A name for the new window to use in the `target` attribute of hyperlink tags
Window features	A list of features and display characteristics for the new window, separated by commas and enclosed together in one set of quotation marks

Special-Purpose Windows

As you've seen in the previous section, one purpose of a new window is to provide users a method to view pages that are not a part of your Web site, without losing them from your site. The easiest way to do this is to provide a complete browser to the user. This is essentially the same as selecting File ➪ New Browser Window (or the equivalent) from your Web browser.

But, what if you just wanted to show a series of images? What if you wanted the user to enter a place where it was necessary for them to only click on hyperlinks to navigate, and not use any forward, back, or address-bar features of the browser?

By using the window features options of the `window.open()` method, you can include all or none of the basic features of a browser window

You can control any of the properties of the window when it's created — absence or presence of toolbars, scrollbars, statusbars, and menus; width and height; and the ability to resize the window. See Table 50-2 for a summary of these properties.

Table 50-2
Window Properties

Property	*Value type*	*Controls*
width	number	Width of the window (in pixels).
height	number	Height of the window (in pixels).
toolbar	yes/no	Include a standard browser toolbar (forward, back, refresh, home, and so on).
menubar	yes/no	Include a standard browser menubar (file, edit, view, tools, and so on). Display the browser statusbar (hyperlink location, security status, page-loading information).
scrollbars	yes/no	Include vertical and horizontal scrollbars as needed to display larger pages.
resizable	yes/no	Include the ability for the user to resize the window.

All or any combination of the properties in Table 50-2 can be included in the third value of the `window.open()` method. For example, to open a new window with the page `index.html` called *IndexWindow*, and with scrollbars and a statusbar but no menu or toolbar, you would use a line like this:

```
Window.open("index.html", "IndexWindow", "toolbar=no,
menubar=no, status=yes, scrollbars=yes")
```

Note that the properties controlling the appearance and features of the window are separated by commas within *one* set of quotation marks.

Specifying the Windows features results in a strange side effect in JavaScript. If just one feature is listed, such as `toolbar=yes`, all other features are set to *no* by default. If you want to specifically exclude one feature but include the others, you'll need to include all the features with a *yes*.

Windows by remote

You can only exert a limited amount of control over a window just by using the `target` attribute of the `anchor` tag. By giving a name to the window using the second property of the `window.open()` method, you can direct hyperlinks to load pages in the new window. But, beyond directing hyperlinks, your ability to do things within the new window is limited.

Using JavaScript, however, you have an option to maintain a "handle" on the created window.

```
<SCRIPT language = "javascript">
var winLinks = null;
function go ( url ) {
  winLinks = window.open("", "links", "width=150,height=200");
  winLinks.location.href = url;
  if (winLinks.opener == null) {
        winLinks.opener = window; }
}
</SCRIPT>
```

What's happening here? First, we created a new JavaScript variable called `winLinks`. This is a null, nondescript variable initially. It has a name, but no value. Then, a value is assigned to `winLinks` — a value that happens to be a new window object sized 150×200 pixels. Note that it doesn't have a location yet (the first property of `window.open()` is blank).

After the window object is created, it is assigned a location using the window's `location.href` property. But notice that we're not using the `window` object name anymore. Now we're using the name of the window object we created, called `winLinks`. Because we created it as a window object, it has all the same properties and methods as the traditional object `window` that represents the current window, including the *href* property used to change its contents.

The last `if` line may be a bit confusing. What is the `opener` property? This is created to make sure the remote window knows about its parent. The parent that "opened" the new window is called the *opener*. You can include JavaScript code in the child window, which refers back to its parent, so that the child can direct the parent to open, close, or load new locations, as shown in the next snippet.

```
<SCRIPT language = "javascript">
function goThere ( url ) {
  if (url) {
        window.opener.location.href = url;
  }
}
</SCRIPT>
```

```
<H3>News links</H3>
<A href="javascript:goThere('http://www.cnn.com')">
CNN OnLine</A><BR>
<A href="javascript:goThere('http://www.abcnews.com')">
ABC News</A><BR>
<A href="javascript:goThere('http://www.cnet.com')">
C|net</A><BR>
```

Closing windows

The methods for closing windows are even more simple than opening the window in the first place. Closing the current active window uses the `close` method of the `window` object.

```
window.close();
```

This is easy enough. Now, let's work on closing some remote windows. One of the things I find annoying with some Internet sites are the extra windows (usually with advertising) that pop up during browsing. Even more annoying is the fact that most site developers don't bother to close any of these windows when you leave the site. You're left with a lot of extra windows eating up system resources.

While you can just leave closing the windows to the user, it's also a simple matter to close the window when you're done with it. Remember the earlier script to create a remote window by creating a window object:

```
<SCRIPT language = "javascript">
var winLinks = null;
function go ( url ) {
  winLinks = window.open("", "links", "width=150,height=200");
  winLinks.location.href = url;
  if (winLinks.opener == null) {
        winLinks.opener = window; }
}
</SCRIPT>
```

In this script, `winLinks` is an object that represents the remote window. Using this object, we can control the remote window in virtually any way, including closing it.

```
winLinks.close();
```

This closes the remote window just as if the user had selected File➪Close or clicked on the close button.

Summary

In HTML and JavaScript, a window is a place where Web pages are displayed. A new window is created in a browser either by selecting File ➪ New Window or by selecting a hyperlink with an external target.

JavaScript recognizes and supports the new windows in a variety of ways. First, it's possible to create a new window from the current window by using the `window` object with the `open` method. Invoking `window.open()` creates a new blank browser window. By adding additional options to the `open` method, it's possible to do much more:

- `window.open("url")` opens a new window and loads the specified page or URL
- `window.open("url", "name")` opens a new window, loads the specified URL, and assigns it a name that can be used with the `target` attribute in a hyperlink.
- `window.open("url", "name", "window features")` opens a new window, loads the specified URL, gives it a name, and determines the use or behavior of one or more window features, such as toolbars, window size, scrollbars, and so on.

By assigning the results of the `window.open` operation to an object, it's possible to control the new window from the originating page.

```
newWindow = window.open();
```

Properties of the object can be used to change the new window, such as the `location` property to load a new URL. Within the new window, a special object called `window.opener` lets the remote window refer back to and control the calling window in the same fashion. In this way, the two windows can interact with one another.

When it's time to close a window, use the `close` method of the appropriate window object. For example, `window.close()` closes the current window, while `newWindow.close()` would close a remote window called *newWindow*.

✦ ✦ ✦