Chapter 24. JavaScript: Adding Dynamic Content to Web Pages

Topics in This Chapter

- Building HTML as the page is loaded
- Monitoring user events
- Building cross-platform scripts
- Mastering JavaScript syntax
- Using JavaScript to customize Web pages
- Using JavaScript to make pages more dynamic
- Using JavaScript to validate CGI forms
- Using JavaScript to manipulate HTTP cookies
- Using JavaScript to interact with and control frames
- Calling Java from JavaScript
- Accessing JavaScript from Java

Despite the similarity in name, JavaScript is very different from Java. JavaScript is a scripting language that is embedded in Web pages and interpreted as the page is loaded. Java is a general-purpose programming language that can be used for desktop application, server-side programs, and applets that execute in a browser. JavaScript can discover a lot of information about the HTML document it is in and can manipulate a variety of HTML elements. Java, if used in a Web page at all, is relatively isolated from the Web page in which it is embedded. JavaScript has no graphics library, explicit threads, or networking. Java has robust graphics (AWT, Swing, Java 2D), an extensive threading library, and networking options that include sockets, RMI, and JDBC.

Currently, there are six versions of JavaScript, 1.0 through 1.5, with a seventh version, JavaScript 2.0, proposed by The Mozilla Organization (http://www.mozilla.org/). JavaScript 1.0 was originally developed by Netscape and released in Netscape Navigator 2.0. Afterwards, Netscape submitted JavaScript to the European Computer Manufacturers Association (ECMA) for standardization. Following that, the ECMA released the ECMAScript standard, ECMA-262, in June 1997. This standard is completely supported by JavaScript 1.1 in Navigator 3.0x. Netscape later departed from the ECMA standard and released JavaScript 1.2 in Navigator 4.0-4.05. In August 1998, the ECMA wrote a second edition of ECMA-262 to align with the corresponding ISO/IEC 16262 standard. The ECMAScript standard, second edition, is implemented as JavaScript 1.3 in Navigator 4.06 and later. Netscape also added significant changes to some of the object classes in JavaScript 1.3 and introduced floating-point constants like `Infinity` and `NaN`. The subsequent version, JavaScript 1.4, provides full support for handling exceptions. To incorporate exceptions and to better handle strings for internationalization in the standard, the ECMA released a third edition of EMCA-262 in December of 1999. The latest browser version from Netscape, Navigator 6, introduced JavaScript 1.5 for full compliance with the third edition of ECMAScript.

To complicate the issue, Microsoft defines its own implementation of the ECMAScript standard, known as JScript. Since Microsoft provides other scripting platforms (Internet Information Server, Windows Scripting Host, and Visual Studio) that support various versions of JScript, we simply summarize which versions of JavaScript are supported by Internet Explorer. For the most part, JavaScript 1.1 is supported in Internet Explorer 3.0x. JavaScript 1.2 is fully supported in Internet Explorer 4.0x and complies also with EMCA-262, second edition. Internet Explorer 5.0x supports JavaScript 1.3 and most of JavaScript 1.4. Internet Explorer 5.5 fully supports the third edition of the EMCA-262 standard and is compatible with JavaScript 1.5.

JavaScript's core language has not changed significantly since version 1.2 and is supported by most popular browsers (Netscape 4 and Internet Explorer 4 and greater). In the following two chapters, we focus on the core language as found in JavaScript 1.2. We do not cover the Document Object Model (DOM) for implementing DHTML. For details on DOM and various scripting versions, we refer you to the following web sites.

ECMAScript Language Specification

http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM

Netscape JavaScript Developer Central

http://developer.netscape.com/tech/javascript/

Microsoft JScript

http://msdn.microsoft.com/scripting/

The Mozilla Organization

http://www.mozilla.org/js/

Document Object Model

http://www.w3.org/DOM/

Two complete reference sources on JavaScript are *Pure JavaScript* by R. Allen Wyke, Jason D. Gilliam, and Charlton Ting (covers JavaScript 1.4), and *Dynamic HTML, The Definitive Reference* by Danny Goodman (covers JavaScript 1.2).

There are two basic ways to use JavaScript in your Web pages. The first is to build HTML dynamically as the Web page is loaded. The second is to monitor various user events and to take action when these events occur. These two syntactic styles are described in the first two sections, with the following section summarizing some other important syntax. These two styles can be combined in a variety of ways and are used for seven general classes of applications: customizing Web pages, making pages more dynamic, validating CGI forms, manipulating cookies, interacting with frames, calling Java from JavaScript, and accessing JavaScript from Java. The remaining sections of this chapter describe each of these application areas, providing two or three examples of each. The following chapter gives details of the standard JavaScript objects in JavaScript 1.2.

24.1 Generating HTML Dynamically

JavaScript code contained inside a `SCRIPT` element is executed as the page is loaded, with any output the code generates being inserted into the document at the place the `SCRIPT` occurred. Listing 24.1 outlines the basic format. Don't worry about all the syntactic details for now; we explain them at the end of this section. For now, just note the standard form.

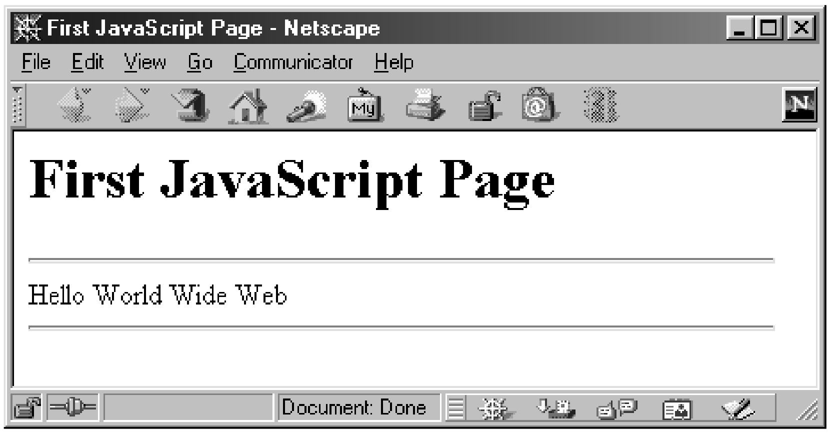
Listing 24.1 Template for Generating HTML with JavaScript

```
...
<BODY>
Regular HTML

<SCRIPT TYPE="text/javascript">
<!--
Build HTML Here
// -->
</SCRIPT>

More Regular HTML
</BODY>
```

You can also use the `SRC` attribute of `SCRIPT` to load remote JavaScript code.

The simplest way to build HTML is to use `document.write`, which places a single string in the current document. Listing 24.2 gives an example, with the result shown in Figure 24-1.

Figure 24-1. The HR elements and the text in between them are generated by JavaScript.


First JavaScript Page - Netscape
File Edit View Go Communicator Help
First JavaScript Page
Hello World Wide Web
Document: Done

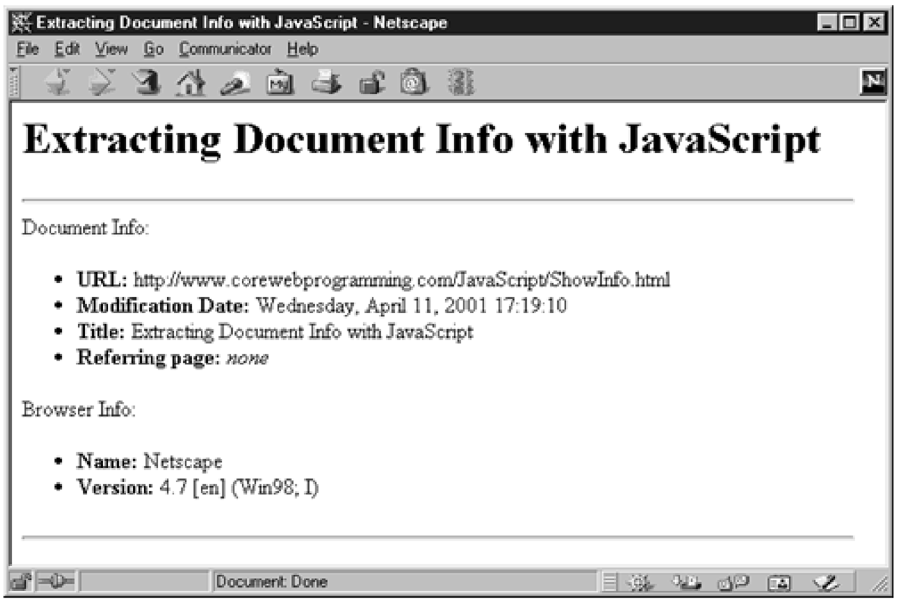
Listing 24.2 `FirstScript.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
  <TITLE>First JavaScript Page</TITLE>
</HEAD>
<BODY>
<H1>First JavaScript Page</H1>

<SCRIPT TYPE="text/javascript">
<!--
document.write("<HR>");
document.write("Hello World Wide Web");
document.write("<HR>");
// -->
```

```
</SCRIPT>

</BODY>
</HTML>
```

Now, this script is not particularly useful because JavaScript did not contribute anything that couldn't have been done with static HTML. It is more common to use JavaScript to build different HTML in different circumstances. Listing 24.3 gives an example, with Figures24-2 and 24-3 showing the results in Netscape Navigator and Microsoft Internet Explorer, respectively. Note the use of the `referringPage` helper function and the "+" string concatenation operator. Building one large string for each chunk of HTML often yields code that is easier to read than it would be with a separate `document.write` for each line of HTML. Also note that this script outputs a linefeed after each line of HTML, using `document.writeln` instead of `document.write` and adding `\n` to each line of text. This linefeed has no effect whatsoever on the resultant Web page. However, some browser versions show the script results when displaying the "source" of a page, and adding the extra newlines makes the result much easier to read. This technique is a very useful debugging tool. If your browser shows you the source in this manner, you can cut and paste the results into a file, then check the syntax of this result with a standard HTML validator (see Section 1.3, "Steps to Publish a Document on the Web"). Highly recommended!

Figure 24-2. `ShowInfo` result in Netscape 4.7 on Windows 98.


Extracting Document Info with JavaScript - Netscape
File Edit View Go Communicator Help
Document: Done

Extracting Document Info with JavaScript

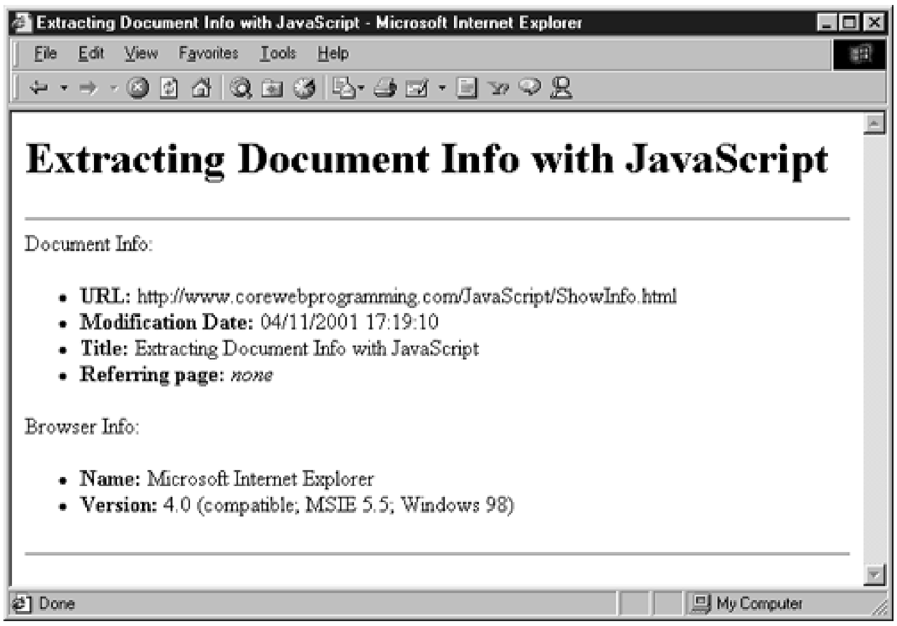
Document Info:

- **URL:** http://www.corewebprogramming.com/JavaScript/ShowInfo.html
- **Modification Date:** Wednesday, April 11, 2001 17:19:10
- **Title:** Extracting Document Info with JavaScript
- **Referring page:** *none*

Browser Info:

- **Name:** Netscape
- **Version:** 4.7 [en] (Win98; I)

Figure 24-3. `ShowInfo` result in Internet Explorer 5.0 on Windows 98.


Extracting Document Info with JavaScript - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

Extracting Document Info with JavaScript

Document Info:

- **URL:** http://www.corewebprogramming.com/JavaScript/ShowInfo.html
- **Modification Date:** 04/11/2001 17:19:10
- **Title:** Extracting Document Info with JavaScript
- **Referring page:** *none*

Browser Info:

- **Name:** Microsoft Internet Explorer
- **Version:** 4.0 (compatible; MSIE 5.5; Windows 98)

Core Approach

If your browser shows you the text and markup resulting from your scripts, verify the syntax with a standard HTML validator.

Listing 24.3 `ShowInfo.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Extracting Document Info with JavaScript</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">
<H1>Extracting Document Info with JavaScript</H1>
<HR>

<SCRIPT TYPE="text/javascript">
<!--

function referringPage() {
  if (document.referrer.length == 0) {
    return("<I>none</I>");
  } else {
    return(document.referrer);
  }
}
document.writeln
  ("Document Info:\n" +
   "<UL>\n" +
   "  <LI><B>URL:</B> " + document.location + "\n" +
   "  <LI><B>Modification Date:</B> " + "\n" +
          document.lastModified + "\n" +
   "  <LI><B>Title:</B> " + document.title + "\n" +
   "  <LI><B>Referring page:</B> " + referringPage() + "\n" +
   "</UL>");
```

```
document.writeln
  ("Browser Info:" + "\n" +
   "<UL>" + "\n" +
   "  <LI><B>Name:</B> " + navigator.appName + "\n" +
   "  <LI><B>Version:</B> " + navigator.appVersion + "\n" +
   "</UL>");

// -->
</SCRIPT>

<HR>
</BODY>
</HTML>
```

Compatibility with Multiple Browsers

Note that the JavaScript code is enclosed inside an HTML comment. This approach is not required but is a good standard practice. Older browsers that do not support JavaScript will automatically ignore the `<SCRIPT>` and `</SCRIPT>` tags but will still see text in between. Using the HTML comment will hide the script contents as well. This works because JavaScript treats both "`//`" and "`<!--`" as the beginning of a single-line comment. Now, this script-hiding strategy is not foolproof, because there are things inside the script that could fool older browsers if you are not careful to avoid them. For instance, in HTML 2.0, the official comment syntax is that comments must be inside pairs of "`--`", which in turn must be between "`<!`" and "`>`". Thus,

```
<!-- Foo -- -- Bar -->
```

is a legal comment, but

```
<!-- Foo -- Bar -->
```

is illegal. Consequently, both of the following are illegal comments in HTML 2.0.

```
<!--
var x = 3;
if (x-->2) // Illegal
  doOneThing();
else
  doAnotherThing();
// -->
<!--
var x = 3;
var y = x--; // Illegal
// -->
```

Surprisingly, JavaScript does not have a property corresponding to "the current language version," so you have to create your own, as in the following example.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
languageVersion = "1.0";
// -->
</SCRIPT>

<SCRIPT LANGUAGE="JavaScript1.1">
```

```
<!--
languageVersion = "1.1";
// -->
</SCRIPT>

...

<SCRIPT LANGUAGE="JavaScript1.4">
<!--
languageVersion = "1.4";
// -->
</SCRIPT>

<SCRIPT LANGUAGE="JavaScript1.5">
<!--
languageVersion = "1.5";
// -->
</SCRIPT>
```

When performing this test do not include the attribute `TYPE="text/javascript"`. In most browsers, adding the `TYPE` attribute results in the script executing regardless of the true JavaScript version. Netscape 6 does not suffer from this problem.

24.2 Monitoring User Events

In addition to building HTML on-the-fly, JavaScript can attach expressions to various HTML elements to be triggered when certain user actions are performed. You can monitor events such as a user clicking on a button or hypertext link, loading or unloading (exiting) a page, moving the mouse on or off a link, giving or taking away the input focus from a `FORM` element, submitting a CGI form, and getting an error when an image is loaded. Listing 24.4 gives an example where the `dontClick` method is attached to a button by the `onClick` attribute. Figures24-4 and 24-5 show the result. The `BUTTON` input element was added to HTML just for JavaScript, but this type of event handler can be attached to a variety of standard HTML elements. Note that `dontClick` was defined in the `HEAD` instead of the `BODY`. This practice is common for functions that don't directly generate HTML.

Figure 24-4. The `DontClick` page before the button is pressed.

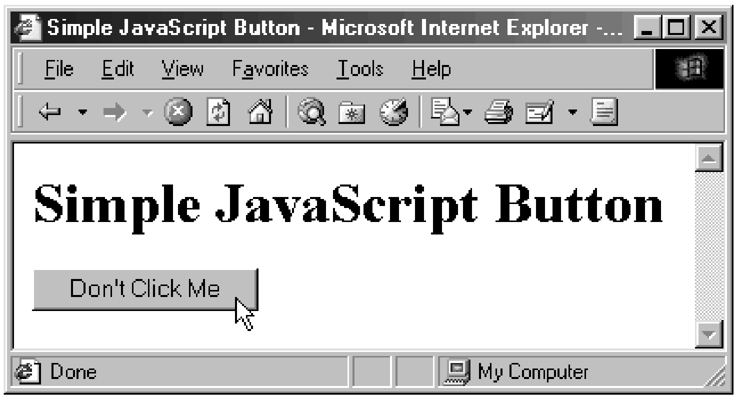
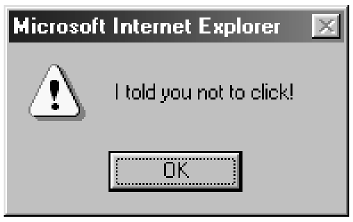

Simple JavaScript Button - Microsoft Internet Explorer -...
File Edit View Favorites Tools Help
Simple JavaScript Button
Don't Click Me
Done
My Computer

Figure 24-5. The result of clicking the button in the `DontClick` page.



I told you not to click!

Listing 24.4 `DontClick.html`

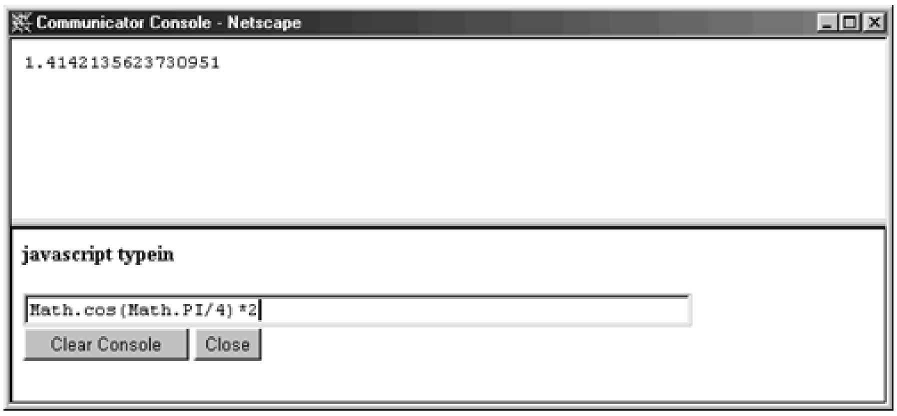
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Simple JavaScript Button</TITLE>

<SCRIPT TYPE="text/javascript">
<!--
function dontClick() {
  alert("I told you not to click!");
}
// -->
</SCRIPT>

</HEAD>
<BODY BGCOLOR="WHITE">
<H1>Simple JavaScript Button</H1>

<FORM>
  <INPUT TYPE="BUTTON"
         VALUE="Don't Click Me"
         onClick="dontClick()">
</FORM>

</BODY>
</HTML>
```

24.3 Mastering JavaScript Syntax

The fundamental syntax of JavaScript looks a lot like Java or C. Most simple constructs will look familiar: `if`, "? :", `while`, `for`, `break`, and `continue` are used just as in the Java programming language. JavaScript 1.2 added a `switch` statement that looks very similar to Java's `switch`, with the exception that the `case` values need not be integers. The operators `+` (addition and string concatenation), `-`, `*`, `/`, `++`, `--`, `&&`, `||`, and so forth are virtually identical. Trailing semicolons are optional, but we use them throughout for the sake of familiarity. We outline several important features here; details are given in the next chapter (JavaScript Quick Reference).

Also note that Netscape provides a convenient interactive JavaScript input window (see Figure 24-6). To open the window, simply enter a URL of "`javascript:`" (nothing after the colon). Alternatively, you enter the complete expression for the URL, for example, "`javascript:Math.cos(Math.PI/4)*2`"; the result is displayed in the browser window. The second approach works both in Netscape and Internet Explorer.

Figure 24-6. Netscape provides an interactive JavaScript "listener."



```
1.4142135623730951
```

```
Math.cos(Math.PI/4)*2
```

Dynamic Typing

The most striking difference between Java and JavaScript is the lack of declared types in JavaScript. You don't declare types for local variables, instance variables (called "properties" in JavaScript lingo), or even return types for functions. A variable can even change type during its lifetime. So, for example, the following is perfectly legal.

```
var x = 5; // int
x = 5.5; // float
x = "five point five"; // String
```

Function Declarations

Functions are declared with the `function` reserved word. The return value is not declared, nor are the types of the arguments. Here are some examples.

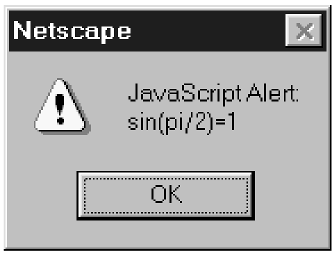
```
function square(x) {
  return(x * x);
}

function factorial(n) {
  if (n <= 0) {
    return(1);
  } else {
    return(n * factorial(n - 1));
  }
}

function printHeading(message) {
  document.writeln("<H1>" + message + "</H1>");
}
```

Functions can be passed around and assigned to variables, as follows:

```
var fun = Math.sin;
alert("sin(pi/2)=" + fun(Math.PI/2));
```

Figure 24-7 shows the result.

Figure 24-7. JavaScript lets you assign a function to a new name.



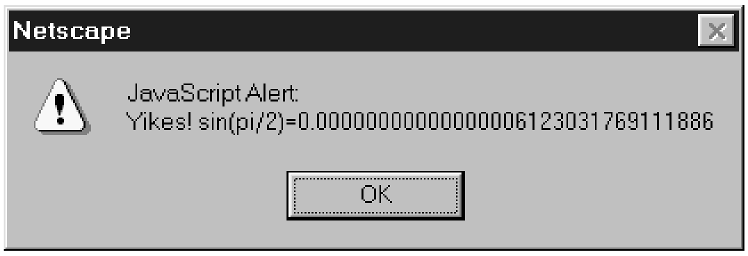
JavaScript Alert:
sin(pi/2)=1

You can also reassign existing functions. In fact, you can even override system functions, as in the following example, although you almost always want to avoid doing that in real applications.

```
Math.sin = Math.cos;  // Don't do this at home
alert("Yikes! sin(pi/2)=" + Math.sin(Math.PI/2));
```

Figure 24-8 shows the result.

Figure 24-8. JavaScript even lets you reassign standard functions.



JavaScript Alert:
Yikes! sin(pi/2)=0.0000000000000000612303176911886

Objects and Classes

JavaScript's approach to object-oriented programming seems a bit haphazard compared to the strict and consistent approach of the Java programming language. Following are a few of the most unusual features.

Fields Can Be Added On-the-Fly

Adding a new property (field) is a simple matter of assigning a value to one. If the field doesn't already exist when you try to assign to it, JavaScript creates it automatically. For instance:

```
var test = new Object();
test.field1 = "Value 1"; // Create field1 property
test.field2 = 7; // Create field2 property
```

Although this approach simplifies the addition of new properties, it also makes it difficult to catch typos because misspelled property names will be happily accepted. Also, if you try to look up a property that doesn't exist, you will get the special `undefined` value. This value compares == to `null`.

You Can Use Literal Notation

You can create objects by using a shorthand "literal" notation of the form

```
{ field1:val1, field2:val2, ... , fieldN:valN }
```

For example, the following gives equivalent values to `object1` and `object2`.

```
var object1 = new Object();
object1.x = 3;
object1.y = 4;
object1.z = 5;
```

```
var object2 = { x:3, y:4, z:5 };
```

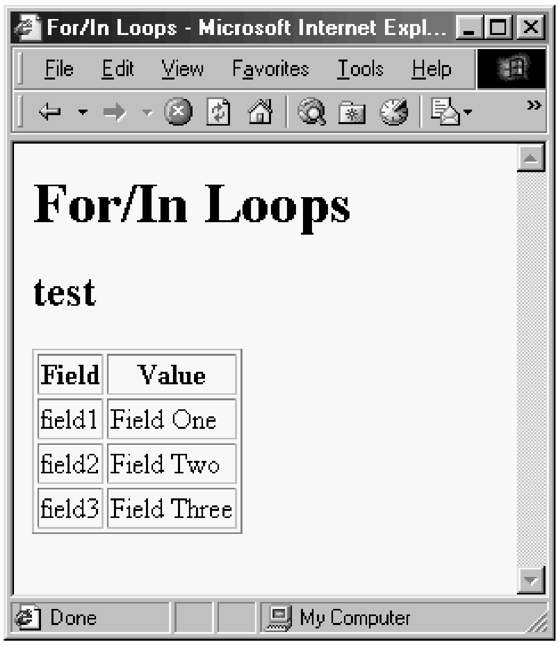
The for/in Statement Iterates over Properties

JavaScript, unlike Java or C++, has a construct that lets you easily retrieve all of the fields of an object. The basic format is as follows:

```
for(fieldName in object) {
  doSomethingWith(fieldName);
}
```

Given a field name, you can access the field through `object["field"]` as well as through `object.field`. This feature is useful when you are iterating over fields in an object, as in Listing 24.5, which defines a general-purpose `makeObjectTable` function that will create an HTML table for a given object. Figure 24-9 gives the result in Internet Explorer 5.0.

Figure 24-9. The `for/in` statement iterates over the properties of an object.


For/In Loops - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Done
My Computer

For/In Loops

test

Field	Value
field1	Field One
field2	Field Two
field3	Field Three

Listing 24.5 `ForIn.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>For/In Loops</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function makeObjectTable(name, object) {
  document.writeln("<H2>" + name + "</H2>");
  document.writeln("<TABLE BORDER=1>\n" +
                   "  <TR><TH>Field<TH>Value");
  for(field in object) {
    document.writeln ("  <TR><TD>" + field +
                      "<TD>" + object[field]);
  }
  document.writeln("</TABLE>");
}
// -->
</SCRIPT>

</HEAD>
<BODY BGCOLOR="WHITE">
<H1>For/In Loops</H1>

<SCRIPT TYPE="text/javascript">
<!--

var test = new Object();
test.field1 = "Field One";
test.field2 = "Field Two";
test.field3 = "Field Three";
makeObjectTable("test", test);

// -->
</SCRIPT>

</BODY>
</HTML>
```

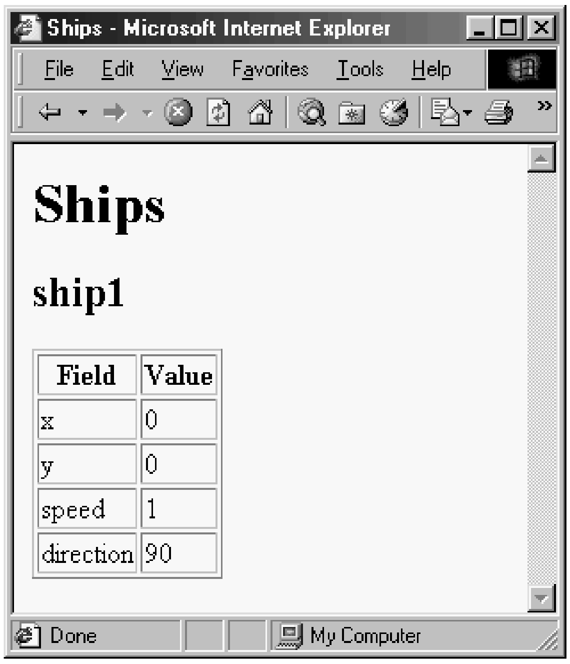
A Constructor Is Just a Function That Assigns to "this"

JavaScript does not have an exact equivalent to Java's class definition. The closest you can get is to define a function that assigns values to properties in the `this` reference. Calling that function with `new` binds `this` to a new `Object`. For example, following is a simple constructor for a `Ship` class.

```
function Ship(x, y, speed, direction) {
  this.x = x;
  this.y = y;
  this.speed = speed;
  this.direction = direction;
}
```

Given the previous definition of `makeObjectTable`, putting the following in a script in the `BODY` of a document yields the result shown in Figure 24-10.

Figure 24-10. Constructors are simply a shorthand way to define objects and assign properties.


Ships - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

Ships

ship1

Field	Value
x	0
y	0
speed	1
direction	90

```
var ship1 = new Ship(0, 0, 1, 90);
makeObjectTable("ship1", ship1);
```

Methods Are Function-Valued Properties

There is no special syntax for defining methods of objects. Instead, you simply assign a function to a property. For instance, here is a version of the `Ship` class that includes a `move` method.

```
function degreesToRadians(degrees) {
  return(degrees * Math.PI / 180.0);
}

function move() {
  var angle = degreesToRadians(this.direction);
  this.x = this.x + this.speed * Math.cos(angle);
  this.y = this.y + this.speed * Math.sin(angle);
}

function Ship(x, y, speed, direction) {
  this.x = x;
  this.y = y;
  this.speed = speed;
  this.direction = direction;
  this.move = move;
}
```

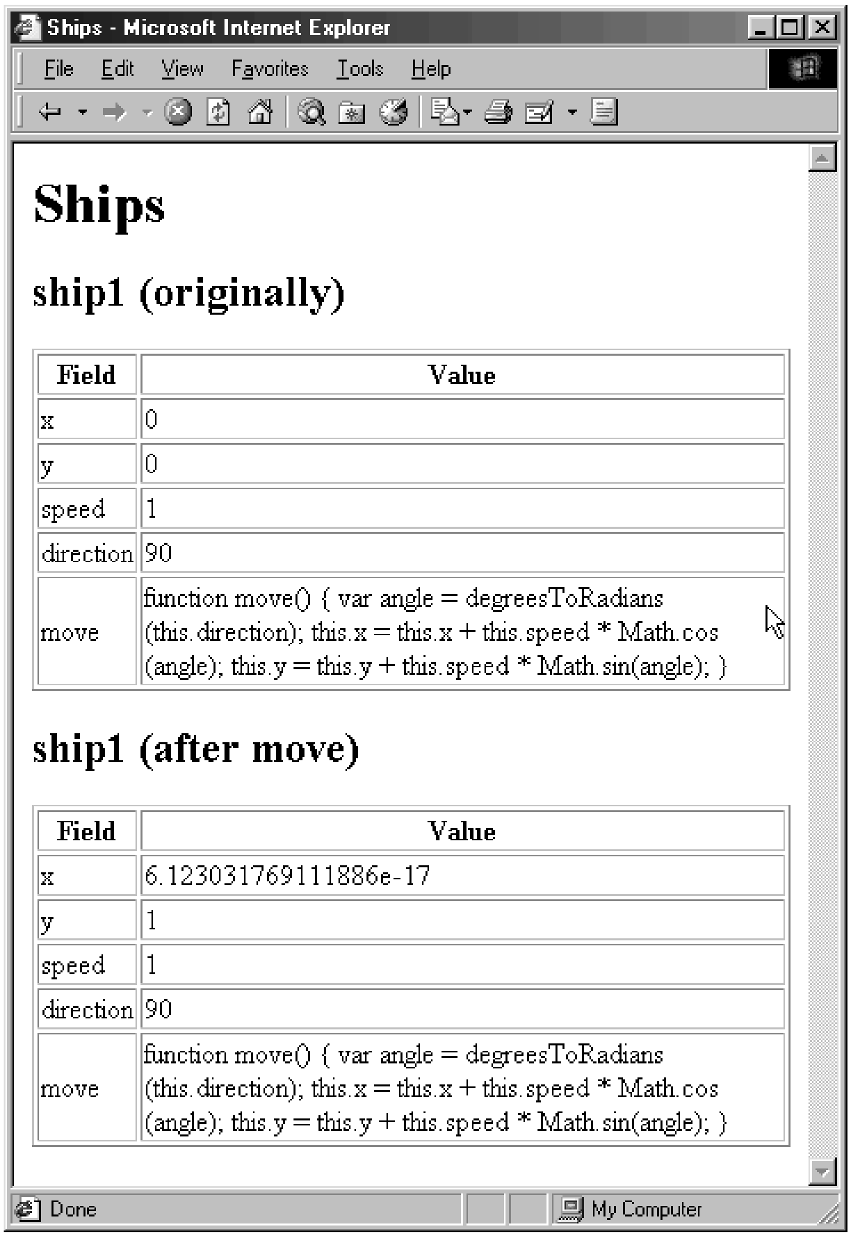
Note the use of `var` before the `angle` variable. This declaration is JavaScript's indication of a local variable. If you forget the declaration, JavaScript won't complain but will treat the variable as a property of the current window. In this case, like-named variables in different functions could conflict with each other. As a result, this behavior of undeclared local variables can lead to hard-to-diagnose problems, so a good standard practice is to always declare your variables.

Core Approach

Introduce all local variables with `var`.

Here is an example that uses the above functions, with the result shown in Figure 24-11.

Figure 24-11. Methods are really a special type of property.


Ships - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

Ships

ship1 (originally)

Field	Value
x	0
y	0
speed	1
direction	90
move	function move() { var angle = degreesToRadians (this.direction); this.x = this.x + this.speed * Math.cos (angle); this.y = this.y + this.speed * Math.sin(angle); }

ship1 (after move)

Field	Value
x	6.123031769111886e-17
y	1
speed	1
direction	90
move	function move() { var angle = degreesToRadians (this.direction); this.x = this.x + this.speed * Math.cos (angle); this.y = this.y + this.speed * Math.sin(angle); }

```
var ship1 = new Ship(0, 0, 1, 90);
makeObjectTable("ship1 (originally)", ship1);
ship1.move();
makeObjectTable("ship1 (after move)", ship1);
```

The prototype Property

You can simplify the creation of methods and constant properties by use of the special `prototype` property. Once at least one object of a given class exists, assigning values to a field in the object stored in the `prototype` property of the class object (really the function object named for the class) gives a shared reference to this value to all members of the class that do not override it. For instance, here is a definition of `Ship` that adds a shared `maxSpeed` property intended to specify the highest speed at which *any* `Ship` can travel.

```
function Ship(x, y, speed, direction) {
  this.x = x;
  this.y = y;
  this.speed = speed;
  this.direction = direction;
}

new Ship(0, 0, 0, 0);
Ship.prototype.move = move;
Ship.prototype.maxSpeed = 50;
```

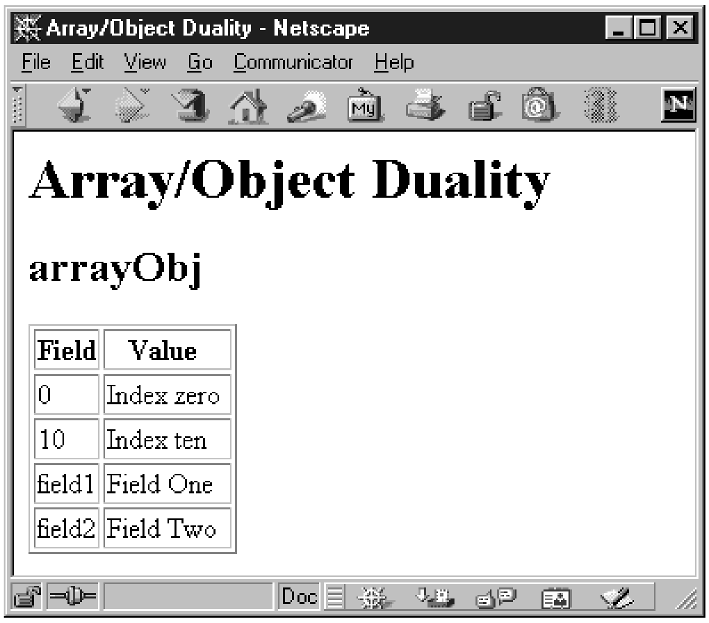
Arrays

For the most part, you can use arrays in JavaScript much like you use Java arrays. Here are a few examples of using the `Array` constructor to simplify the building of arrays.

```
var squares = new Array(5);
for(var i=0; i<squares.length; i++) {
  vals[i] = i * i;
}
// Or, in one fell swoop:
var squares = new Array(0, 1, 4, 9, 16);
var array1 = new Array("fee", "fie", "fo", "fum");
// Literal Array notation for creating an array.
var array2 = [ "fee", "fie", "fo", "fum" ];
```

Using arrays this way is probably the simplest use. Behind the scenes, however, JavaScript simply represents arrays as objects with numbered fields. You can access named fields by using either `object.field` or `object["field"]`, but you access numbered fields only by `object[fieldNumber]`. Here is an example, with Figure 24-12 showing the result.

Figure 24-12. Arrays are really just objects with numbered fields.


Array/Object Duality - Netscape
File Edit View Go Communicator Help
Doc

Array/Object Duality

arrayObj

Field	Value
0	Index zero
10	Index ten
field1	Field One
field2	Field Two

```
var arrayObj = new Object();
arrayObj[0] = "Index zero";
arrayObj[10] = "Index ten";
arrayObj.field1 = "Field One";
arrayObj["field2"] = "Field Two";
```

```
makeObjectTable("arrayObj", arrayObj);
```

In general, it is a good idea to avoid mixing array and object notation and to treat them as two separate varieties of object. Occasionally, however, it is convenient to mix the notations because the array notation is required when an existing string is to be used as a property name. For instance, the `makeObjectTable` (Listing 24.5) function relied upon this capability.

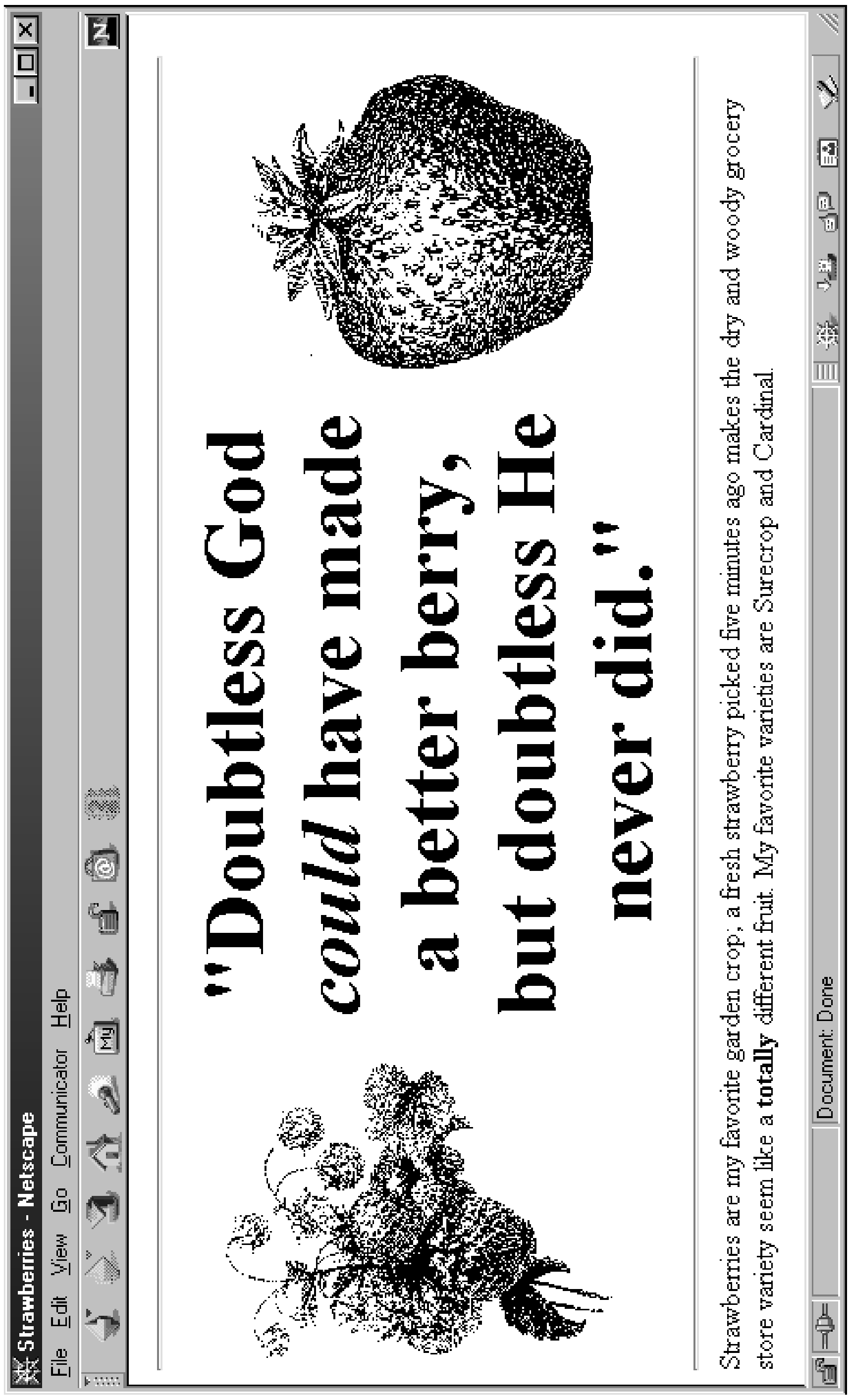
24.4 Using JavaScript to Customize Web Pages

Because JavaScript can determine several characteristics of the current browser and document, you can use it to build different Web pages in different circumstances. For instance, you might want to use certain Microsoft-specific features on Internet Explorer and Netscape-specific features on Netscape Navigator. Or omit certain background information if users came to your page from another one of your pages, as opposed to an outside page. Or use smaller images if the user has a small screen. Or use embedded objects only if the browser has a plug-in that supports them. Integration with Java (Section 24.9) will allow additional applications such as printing only a "Sorry, no Web page for you, buddy!" page to users who visit from spam-tolerant ISPs, or removing the prominent link to your resumé when people access your Web page from your company domain. Following are two examples. The first shows how to customize the page for the browser window size, and the second illustrates how to tell if certain plug-ins are available.

Adjusting to the Browser Window Size

Netscape 4.0 introduced the `window.innerWidth` and `window.innerHeight` properties, which let you determine the usable size of the current browser window. Listing 24.6 uses this value to shrink/stretch images to a fixed percentage of the window width and to adjust the size of a heading font accordingly. Figures24-13 and 24-14 show the results in a large and a small browser window.

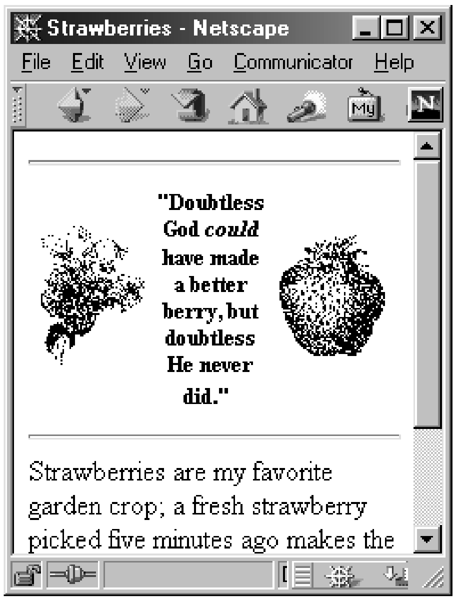
Figure 24-13. In a large browser, large images and fonts are used.



"Doubtless God *could* have made a better berry, but doubtless He never did."

Strawberries are my favorite garden crop; a fresh strawberry picked five minutes ago makes the dry and woody grocery store variety seem like a **totally** different fruit. My favorite varieties are Surecrop and Cardinal.

Figure 24-14. Even in a tiny browser window, the heading is still legible.



"Doubtless God *could* have made a better berry, but doubtless He never did."

Strawberries are my favorite garden crop; a fresh strawberry picked five minutes ago makes the

There are a couple of other stylistic notes to make about this example. First of all, notice that the function definitions are placed in the `HEAD`, even though they are used in the `BODY`. This is a standard practice that has three benefits. (1): It can make the actual script easier to read. (2): It allows a single function to be used multiple places. And (3): Because the `HEAD` is parsed before the `BODY`, JavaScript routines defined in the `HEAD` will be available even if the user interrupts the loading or clicks on an image or cross-reference before the page is done loading. This behavior is particularly valuable for event-handling functions.

Core Approach

Define JavaScript functions in the `HEAD`, especially if they will be used in event handlers.

Also note the use of single quotes instead of double quotes for strings in this example. JavaScript allows either, so using single quotes for `document.writeln` makes it easier to embed double quotes inside the string, as needed in this case for the "better berry" quotation.

Core Note

Either single or double quotes can be used for JavaScript strings. Double quotes can be embedded inside strings created with single quotes; single quotes can be used inside strings made with double quotes.

Listing 24.6 `Strawberries.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Strawberries</TITLE>

<SCRIPT TYPE="text/javascript">
<!--
function image(url, width, height) {
 return('<IMG SRC="' + url + '"' +
            ' WIDTH=' + width +
            ' HEIGHT=' + height + '>');
}

function strawberry1(width) {
  return(image("Strawberry1.gif", width,
Math.round(width*1.323)));
}

function strawberry2(width) {
  return(image("Strawberry2.gif", width,
Math.round(width*1.155)));
}
// -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="WHITE">

<HR>
<SCRIPT TYPE="text/javascript">
<!--
  var imageWidth = window.innerWidth/4;
  var fontSize = Math.min(7, Math.round(window.innerWidth/100));
}

document.writeln
  ('<TABLE>\n' +
   '  <TR><TD>' + strawberry1(imageWidth) + '\n' +
   '      <TH><FONT SIZE=' + fontSize + '>\n' +
   '          "Doubtless God <I>could</I> have made\n' +
   '          a better berry, but doubtless He\n' +
   '          never did."</FONT>\n' +
   '      <TD>'  + strawberry2(imageWidth) + '\n' +
   '</TABLE>');
// -->
</SCRIPT>
<HR>

Strawberries are my favorite garden crop; a fresh
strawberry picked five minutes ago makes the dry and
woody grocery store variety seem like a <B>totally</B>
different fruit. My favorite varieties are Surecrop
and Cardinal.

</BODY>
</HTML>
```

Determining Whether Plug-Ins Are Available

In Netscape (but not Internet Explorer), the `navigator.plugins` array contains information about the available browser plug-ins. Each element in this array is a `Plugin` object that has `name`, `description`, `filename`, and `length` properties and contains an array of `MimeType` objects. These properties give a short name to the plug-in, a textual description, the filename containing the plug-in, and the number of supported MIME types, respectively. Each `MimeType` object has properties `type` (MIME datatype such as "text/html"), `description` (descriptive text), `enabledPlugin` (the `Plugin` object supporting this type), and `suffixes` (a comma-separated list of file extensions associated with this type). An interesting aspect of JavaScript arrays is that you can reference them with a string "index" instead of an integer. In fact, as explained in Section 24.3 (Mastering JavaScript Syntax), arrays in JavaScript are really just objects with numbered fields. Anyhow, you can use this shortcut to determine if a plug-in is installed, as in the following snippet.

```
if (navigator.plugins["Cosmo Player 1.0"]) {
  document.write('<EMBED SRC="coolWorld.vrml" ...>"');
} else {
  document.write('This example requires VRML.');
}
```

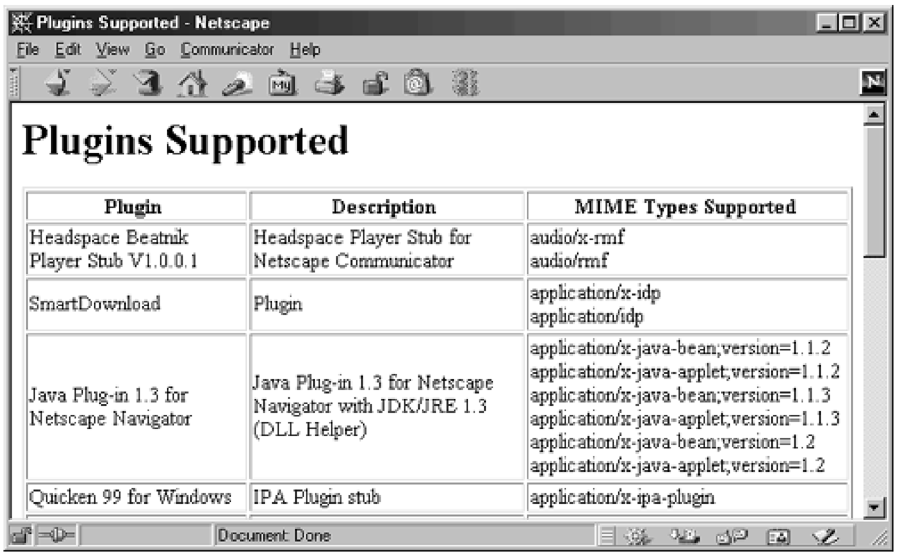
Note that this code tells you whether a *particular* plug-in is available. If you are more concerned about whether a certain MIME type is supported *somehow* (directly, through a plug-in, or through an external application), you can check the `navigator.mimeTypes` property. For example,

```
if (navigator.mimeTypes["application/postscript"]) {
  addPostScriptLink();
}
```

For more information, see Section 25.19 (The MimeType Object).

Listing 24.7 uses this approach to build a table of available plug-ins and their supported MIME types. Figure 24-15 show the results in Netscape 4.7.

Figure 24-15. Partial list of plug-ins available in this version of Netscape 4.7.


Plugins Supported - Netscape
File Edit View Go Communicator Help
Document: Done

Plugins Supported

Plugin	Description	MIME Types Supported
Headspace Beatnik Player Stub V1.0.0.1	Headspace Player Stub for Netscape Communicator	audio/x-rmf audio/rmf
SmartDownload	Plugin	application/x-idp application/idp
Java Plug-in 1.3 for Netscape Navigator	Java Plug-in 1.3 for Netscape Navigator with JDK/JRE 1.3 (DLL Helper)	application/x-java-bean;version=1.1.2 application/x-java-applet;version=1.1.2 application/x-java-bean;version=1.1.3 application/x-java-applet;version=1.1.3 application/x-java-bean;version=1.2 application/x-java-applet;version=1.2
Quicken 99 for Windows	IPA Plugin stub	application/x-ipa-plugin

Listing 24.7 `Plugins.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Plug-ins Supported</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function printRow(plugin) {
  document.write
    ("  <TR><TD>" + plugin.name + "\n" +
     "      <TD>" + plugin.description + "\n" +
     "      <TD>");
  document.write(plugin[0].type);
  for(var i=1; i<plugin.length; i++)
    document.writeln("<BR>" + plugin[i].type);
}

// -->
</SCRIPT>
</HEAD>
<BODY>
<H1>Plug-ins Supported</H1>

<SCRIPT TYPE="text/javascript">
<!--
if (navigator.appName == "Netscape") {
  document.writeln
    ("<TABLE BORDER=1>\n" +
     "  <TR><TH>Plug-in\n" +
     "      <TH>Description\n" +
     "      <TH>MIME Types Supported");
  for(var i=0; i<navigator.plugins.length; i++)
    printRow(navigator.plugins[i]);
  document.writeln
    ("\n</TABLE>");
}
// -->
</SCRIPT>

</BODY>
</HTML>
```

24.5 Using JavaScript to Make Pages Dynamic

In most of the previous examples, parts of the document were built dynamically (when the page was loaded), but the resultant document was normal HTML. JavaScript can also be used to create elements that are dynamic. For instance, one common application is to create images that change when the user moves the mouse over them. This application can be used to implement toolbars with regions that "light up" to indicate hypertext links or custom buttons that show a grayed-out image when you press them. Alternatively, by using timers, JavaScript can animate images even without being triggered by user events. JavaScript can manipulate layers, scroll the document, and even move the browser window around on the screen.

Modifying Images Dynamically

In JavaScript, the `document.images` property contains an array of `Image` objects corresponding to each `IMG` element in the current document. To display a new image, simply set the `SRC` property of an existing image to a string representing a different image file. For instance, the following function changes the first image in a document.

```
function changeImage() {
  document.images[0].src = "images/new-image.gif";
}
```

This function could be invoked from an event handler (e.g., when the user clicks a button) or even executed automatically after a certain amount of time. Now, referring to images by number is not very flexible, because the addition of a new image in the middle of the document would require changing the references to all later images. Fortunately, JavaScript recognizes the `NAME` attribute of the `IMG` element. For example, if you provide a name for your image

```
<IMG SRC="cool-image.jpg" id="cool"
     WIDTH=75 HEIGHT=25>
```

then you could refer to an array element by name instead of number, as in

```
function improveImage() {
  document.images["cool"].src = "way-cool.jpg";
}
```

A Clickable Image Button

The concept of images that change when you click on them is implemented with a clickable image button. For example, following is a `clickButton` function that temporarily changes an image, switching it back to the original version after 1/10 of a second. To do this, it uses the `setImage` and `setTimeout` function. The first of these is defined as follows, where `setTimeout` is a built-in routine that takes a string designating a JavaScript expression and a time in milliseconds. It returns immediately but starts a background process that waits for the specified time, then executes the code specified by the string.

```
function setImage(name, image) {
  document.images[name].src = image;
}

function clickButton(name, grayImage) {
  var origImage = document.images[name].src;
  setImage(name, grayImage);
  var resetString =
    "setImage('" + name + "', '" + origImage + "')";
  setTimeout(resetString, 100);
}
```

To use this routine for a clickable image button, we need to do two more things: (1) attach the routine to a button or buttons, and (2) make sure that the images needed are already cached by the browser. The first step is straightforward: simply use the `onClick` attribute of the `<A HREF...>` element, as shown below.

```
<A HREF="location1.html"
   onClick="clickButton('Button1',
                        'images/Button1-Down.gif')">
<IMG SRC="images/Button1-Up.gif" id="Button1"
     WIDTH=150 HEIGHT=25></A>

<A HREF="location2.html"
   onClick="clickButton('Button2',
                        'images/Button2-Down.gif')">
<IMG SRC="images/Button2-Up.gif" id="Button2"
     WIDTH=150 HEIGHT=25></A>
```

Finally, before trying to display an image, you should make sure it is already loaded. This extra step will prevent long pauses when the button is pressed. You can do this step by creating an `Image` object (Section 25.12), then setting its `SRC` property. Oddly, this `Image` object never actually gets used; its only purpose is to force the browser to load (and cache) the image. Here is an example.

```
imageFiles = new Array("images/Button1-Up.gif",
                       "images/Button1-Down.gif",
                       "images/Button2-Up.gif",
                       "images/Button2-Down.gif");
imageObjects = new Array(imageFiles.length);

for(var i=0; i<imageFiles.length; i++) {
  imageObjects[i] = new Image(150, 25);
  imageObjects[i].src = imageFiles[i];
}
```

Listing 24.8 shows the whole process put together. If you are handling a lot of images, you can simplify the process by having a consistent naming scheme for the images; we give an example later in this section.

Listing 24.8 `ImageButton.html`

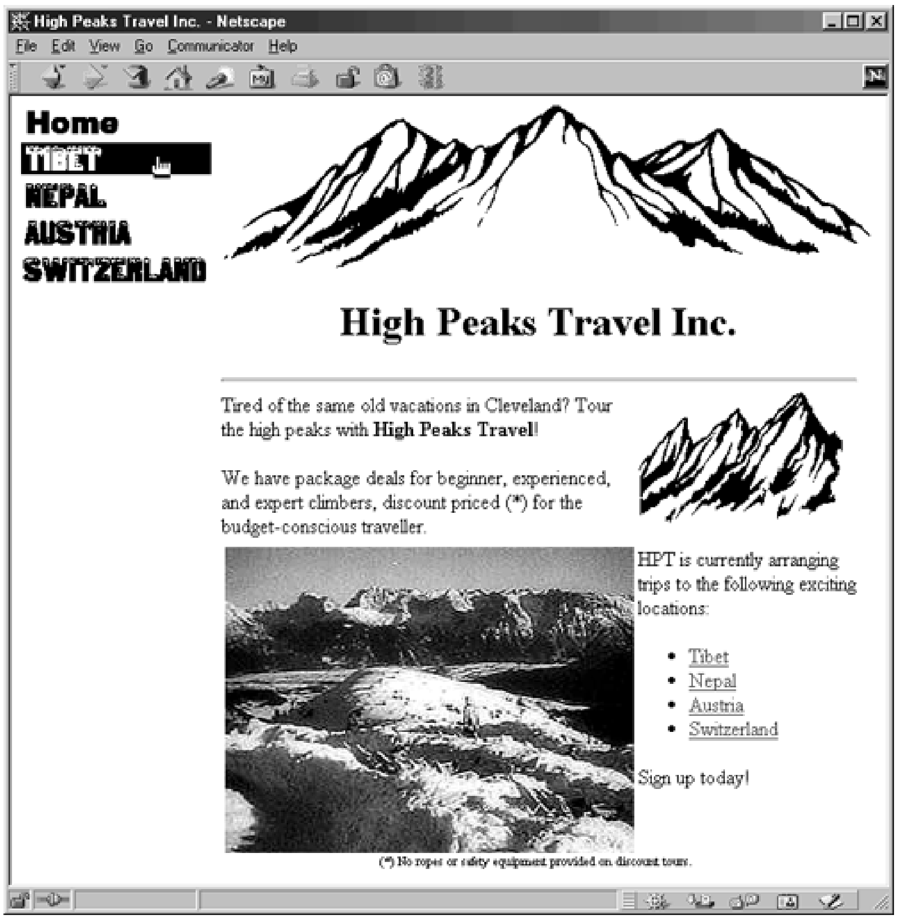
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>JavaScript Image Buttons</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

imageFiles = new Array("images/Button1-Up.gif",
                       "images/Button1-Down.gif",
                       "images/Button2-Up.gif",
                       "images/Button2-Down.gif");
imageObjects = new Array(imageFiles.length);
for(var i=0; i<imageFiles.length; i++) {
  imageObjects[i] = new Image(150, 25);
  imageObjects[i].src = imageFiles[i];
}

function setImage(name, image) {
  document.images[name].src = image;
}

function clickButton(name, grayImage) {
  var origImage = document.images[name].src;
  setImage(name, grayImage);
  var resetString =
    "setImage('" + name + "', '" + origImage + "')";
  setTimeout(resetString, 100);
}

// -->
</SCRIPT>

</HEAD>
<BODY>
```

```
<H1>JavaScript Image Buttons</H1>


<A HREF="location1.html"
   onClick="clickButton('Button1', 'images/Button1-Down.gif')">
<IMG SRC="images/Button1-Up.gif" id="Button1"
     WIDTH=150 HEIGHT=25></A>

<A HREF="location2.html"
   onClick="clickButton('Button2', 'images/Button2-Down.gif')">
<IMG SRC="images/Button2-Up.gif" id="Button2"
     WIDTH=150 HEIGHT=25></A>

</BODY>
</HTML>
```

Highlighting Images Under the Mouse

An even more common application of the image modification process is to create a series of images that change as the user moves the mouse over them, using the hypertext link's `onMouseOver` to display the highlighted image and `onMouseOut` to change the image back. This approach can make toolbars more appealing by providing visual cues about which regions are clickable. However, when you are dealing with a large number of images, listing each explicitly when preloading them can be tedious. Listing 24.9 shows an approach that simplifies this process considerably: using consistent names. The normal image and highlighted image are both derived from the `NAME` of the `IMG` element (see `regularImageFile` and `negativeImageFile`), obviating the need to list the full filenames in the array of images to be preloaded or to pass the highlighted image name in the `onMouseOver` call. Toolbars of this type are most commonly used with frames. Listing 24.10 and Listing 24.11 show the rest of the frame structure; Figure 24-16 shows the results.

Figure 24-16. Toolbar entries light up when you move the mouse over them.


High Peaks Travel Inc. - Netscape
File Edit View Go Communicator Help
Home
TIBET
NEPAL
AUSTRIA
SWITZERLAND

High Peaks Travel Inc.

Tired of the same old vacations in Cleveland? Tour the high peaks with **High Peaks Travel**!

We have package deals for beginner, experienced, and expert climbers, discount priced (*) for the budget-conscious traveller.

HPT is currently arranging trips to the following exciting locations:

- Tibet
- Nepal
- Austria
- Switzerland

Sign up today!

(*) No ropes or safety equipment provided on discount tours.

If you can't remember how to use frames, this would be a good time to review them (Chapter 4), because they are used quite frequently with JavaScript.

Listing 24.9 `HighPeaksNavBar.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>High Peaks Navigation Bar</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

// Given "Foo", returns "images/Foo.gif".

function regularImageFile(imageName) {
  return("images/" + imageName + ".gif");
}

// Given "Bar", returns "images/Bar-Negative.gif".

function negativeImageFile(imageName) {
  return("images/" + imageName + "-Negative.gif");
}

// Cache image at specified index. E.g., given index 0,
// take imageNames[0] to get "Home". Then preload
// images/Home.gif and images/Home-Negative.gif.
```

```
function cacheImages(index) {
  regularImageObjects[index] = new Image(150, 25);
  regularImageObjects[index].src =
    regularImageFile(imageNames[index]);
  negativeImageObjects[index] = new Image(150, 25);
  negativeImageObjects[index].src =
    negativeImageFile(imageNames[index]);
}
imageNames = new Array("Home", "Tibet", "Nepal",
                       "Austria", "Switzerland");
regularImageObjects = new Array(imageNames.length);
negativeImageObjects = new Array(imageNames.length);

// Put images in cache for fast highlighting.
for(var i=0; i<imageNames.length; i++) {
  cacheImages(i);
}

// This is attached to onMouseOver -- change image
// under the mouse to negative (reverse video) version.

function highlight(imageName) {
  document.images[imageName].src =
    negativeImageFile(imageName);
}

// This is attached to onMouseOut -- return image to
// normal.

function unHighlight(imageName) {
  document.images[imageName].src = regularImageFile(imageName);
}

// -->
</SCRIPT>
</HEAD>

<BODY BGCOLOR="WHITE">

<TABLE BORDER=0 WIDTH=150 BGCOLOR="WHITE"
       CELLPADDING=0 CELLSPACING=0>
  <TR><TD><A HREF="Home.html"
             TARGET="Main"
             onMouseOver="highlight('Home')"
             onMouseOut="unHighlight('Home')">
          <IMG SRC="images/Home.gif"
               id="Home"
               WIDTH=150 HEIGHT=25 BORDER=0>
          </A>
  <TR><TD><A HREF="Tibet.html"
             TARGET="Main"
             onMouseOver="highlight('Tibet')"
```

```
            onMouseOut="unHighlight('Tibet')">
         <IMG SRC="images/Tibet.gif"
              id="Tibet"
              WIDTH=150 HEIGHT=25 BORDER=0>
         </A>
  <TR><TD><A HREF="Nepal.html"
            TARGET="Main"
            onMouseOver="highlight('Nepal')"
            onMouseOut="unHighlight('Nepal')">
         <IMG SRC="images/Nepal.gif"
              id="Nepal"
              WIDTH=150 HEIGHT=25 BORDER=0></A>
  <TR><TD><A HREF="Austria.html"
            TARGET="Main"
            onMouseOver="highlight('Austria')"
            onMouseOut="unHighlight('Austria')">
         <IMG SRC="images/Austria.gif"
              id="Austria"
              WIDTH=150 HEIGHT=25 BORDER=0></A>
  <TR><TD><A HREF="Switzerland.html"
            TARGET="Main"
            onMouseOver="highlight('Switzerland')"
            onMouseOut="unHighlight('Switzerland')">
         <IMG SRC="images/Switzerland.gif"
              id="Switzerland"
              WIDTH=150 HEIGHT=25 BORDER=0></A>
</TABLE>
</BODY>
</HTML>
```

Listing 24.10 `HighPeaks.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
  <TITLE>High Peaks Travel Inc.</TITLE>
</HEAD>

<FRAMESET COLS="160,*" FRAMEBORDER=0 BORDER=0>
  <FRAME SRC="HighPeaksNavBar.html" SCROLLING="NO">
  <FRAME SRC="HighPeaksIntro.html" id="Main">

  <NOFRAMES>
    If you can't hack frames, how do you expect
    to handle the Himalayas? Get a real browser.
  </NOFRAMES>
</FRAMESET>

</HTML>
```

Listing 24.11 `HighPeaksIntro.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
```

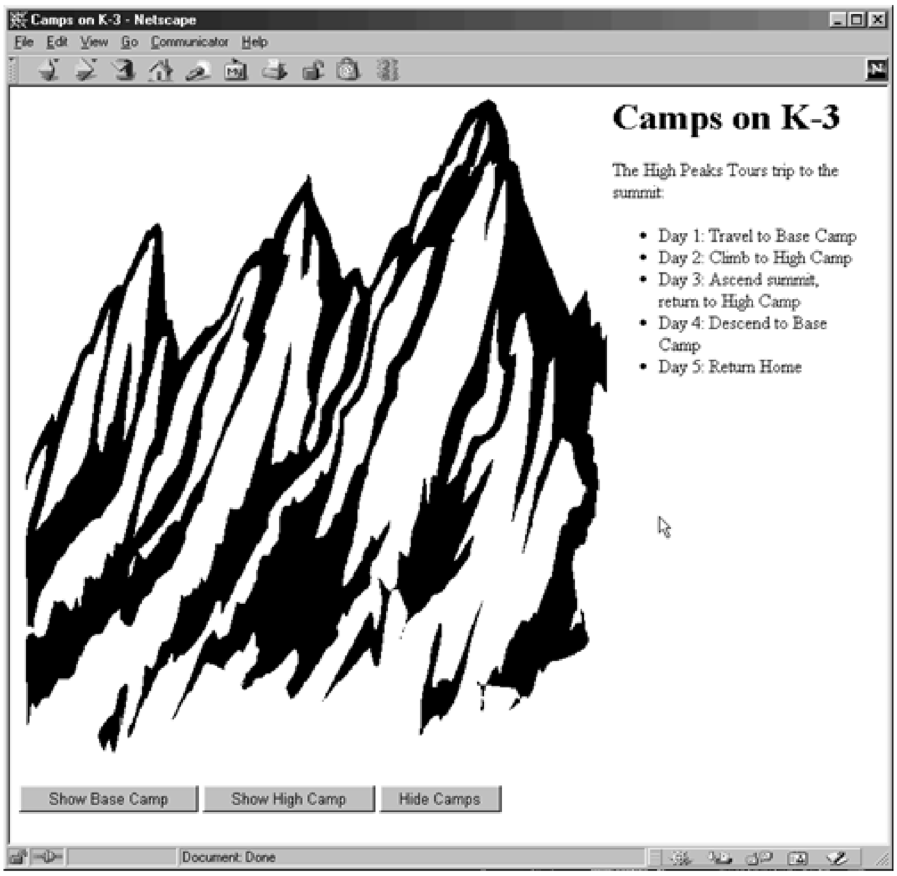
```
<HEAD>
  <TITLE>High Peaks Travel Inc.</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">
<CENTER>
  <IMG SRC="images/peak2.gif" WIDTH=511 HEIGHT=128>
</CENTER>
<H1 ALIGN="CENTER">High Peaks Travel Inc.</H1>
<HR>

<IMG SRC="images/peak1.gif" WIDTH=170 HEIGHT=121 ALIGN="RIGHT">
Tired of the same old vacations in Cleveland?
Tour the high peaks with <B>High Peaks Travel</B>!
<P>
We have package deals for beginner, experienced, and expert
climbers, discount priced (*) for the budget-conscious
traveller.
<BR CLEAR="ALL">
<IMG SRC="images/peak.jpg" WIDTH=320 HEIGHT=240
     ALIGN="LEFT">
HPT is currently arranging trips to the following
exciting locations:
<UL>
  <LI><A HREF="Tibet.html">Tibet</A>
  <LI><A HREF="Nepal.html">Nepal</A>
  <LI><A HREF="Austria.html">Austria</A>
  <LI><A HREF="Switzerland.html">Switzerland</A>
</UL>
Sign up today!

<BR CLEAR="ALL">
<CENTER>
<FONT SIZE="-2">(*) No ropes or safety equipment provided
on discount tours. </FONT>
</CENTER>

</BODY>
</HTML>
```

Moving Layers

Netscape 4.0 introduced "Layers." These are HTML regions that can overlap and be positioned arbitrarily; they are covered in Section 5.12 (Layers). JavaScript 1.2 lets you access layers through the `document.layers` array, each element of which is a `Layer` object with properties corresponding to the attributes of the `LAYER` element. A named layer can be accessed through `document.layers["layer name"]` rather than through an index or simply by use of `document.layerName`. Layers can be accessed this way no matter how they are defined in the HTML: by the `LAYER` element, by the `ILAYER` element, or through style sheets.

Listing 24.12 presents an example with two layers that are initially hidden (Figure 24-17). When a certain button is pressed, the first layer is made visible near the upper-left corner, then moves down over the top of the regular page to its final location, where it annotates an image (Figure 24-18). Clicking a second button hides the first layer and displays a second, which also drifts to its final location. The properties and methods of the `Layer` object are described in Section 25.15, but for this example, the properties of interest are `visibility` (`show` or `hidden`) and `pageX` (absolute location in window). The methods used are `moveToAbsolute` (position layer at absolute location) and `moveBy` (move layer relative to its previous position).

Figure 24-17. When the page is first loaded in Netscape 4.7, both layers are hidden.

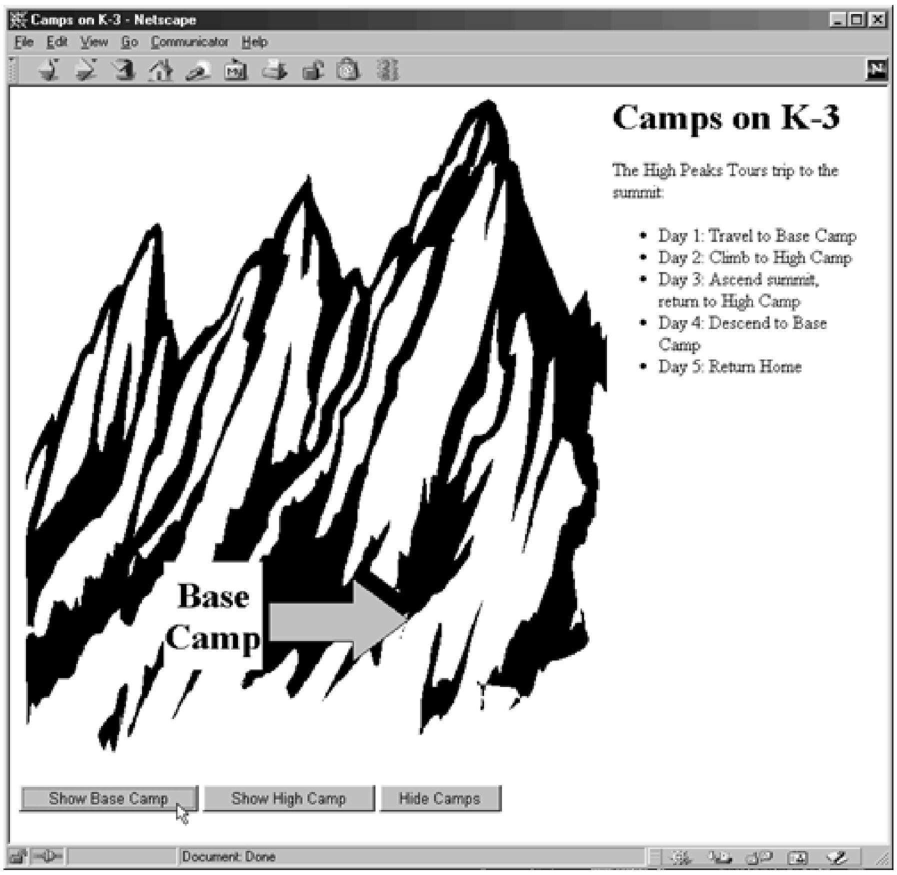


Camps on K-3

The High Peaks Tours trip to the summit:

- Day 1: Travel to Base Camp
- Day 2: Climb to High Camp
- Day 3: Ascend summit, return to High Camp
- Day 4: Descend to Base Camp
- Day 5: Return Home

Figure 24-18. Clicking "Show Base Camp" turns on the base camp layer near the top-left corner of the page. It then drifts down to its final position.



Camps on K-3

The High Peaks Tours trip to the summit:

- Day 1: Travel to Base Camp
- Day 2: Climb to High Camp
- Day 3: Ascend summit, return to High Camp
- Day 4: Descend to Base Camp
- Day 5: Return Home

This example runs only in Netscape 4; LAYERS are not legal in the HTML 4.0 specification and are unsupported in Netscape 6. Furthermore, Internet Explorer uses a different document object model (DOM) to access and move the layers. We provide an on-line, cross-browser version of this example at http://www.corewebprogramming.com/. For detailed information on how to upgrade existing documents containing the LAYER element to Netscape 6 and Internet Explorer, see http://sites.netscape.net/ekrock/standards.html.

Listing 24.12 `Camps.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Camps on K-3</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function hideCamps() {
  // Netscape 4 document model.
  document.layers["baseCamp"].visibility = "hidden";
  document.layers["highCamp"].visibility = "hidden";
  // Or document.baseCamp.visibility = "hidden";
}

function moveBaseCamp() {
  baseCamp.moveBy(1, 3);
  if (baseCamp.pageX < 130) {
    setTimeout("moveBaseCamp()", 10);
  }
}
// Hide camps, position base camp near top-left corner,
// make it visible, then have it slowly drift down to
// final position.

function showBaseCamp() {
  hideCamps();
  baseCamp =  document.layers["baseCamp"];
  baseCamp.moveToAbsolute(0, 20);
  baseCamp.visibility = "show";
  moveBaseCamp();
}

function moveHighCamp() {
  highCamp.moveBy(2, 1);
  if (highCamp.pageX < 110) {
    setTimeout("moveHighCamp()", 10);
  }
}

// Hide camps, position high camp near top-left corner,
// make it visible, then have it slowly drift down to
// final position.

function showHighCamp() {
  hideCamps();
  highCamp =  document.layers["highCamp"];
  highCamp.moveToAbsolute(0, 65);
  highCamp.visibility = "show";
  moveHighCamp();
}

// -->
</SCRIPT>
</HEAD>
<BODY>

<IMG SRC="images/peak4.gif" WIDTH=511 HEIGHT=600 ALIGN="LEFT">
<H1>Camps on K-3</H1>
The High Peaks Tours trip to the summit:
<UL>
  <LI>Day 1: Travel to Base Camp
  <LI>Day 2: Climb to High Camp
  <LI>Day 3: Ascend summit, return to High Camp
  <LI>Day 4: Descend to Base Camp
  <LI>Day 5: Return Home
</UL>
<BR CLEAR="ALL">

<!--           LAYER only supported Netscape 4           -->
<LAYER id="highCamp" PAGEX=50 PAGEY=100 VISIBILITY="hidden">
  <TABLE>
    <TR><TH BGCOLOR="WHITE" WIDTH=50>
```

```
        <FONT SIZE="+2">High Camp</FONT>
        <TD><IMG SRC="images/Arrow-Right.gif">
  </TABLE>
</LAYER>

<!--           LAYER only supported Netscape 4           -->
<LAYER id="baseCamp" PAGEX=50 PAGEY=100 VISIBILITY="hidden">
  <TABLE>
    <TR><TH BGCOLOR="WHITE" WIDTH=50>
        <FONT SIZE="+3">Base Camp</FONT>
        <TD><IMG SRC="images/Arrow-Right.gif">
  </TABLE>
</LAYER>

<FORM>
  <INPUT TYPE="Button" VALUE="Show Base Camp"
         onClick="showBaseCamp()">
  <INPUT TYPE="Button" VALUE="Show High Camp"
         onClick="showHighCamp()">
  <INPUT TYPE="Button" VALUE="Hide Camps"
         onClick="hideCamps()">
</FORM>

</BODY>
</HTML>
```

24.6 Using JavaScript to Validate HTML Forms

Another important application of JavaScript is to check the format of form fields before the form is submitted to the server. Contacting the server can be expensive, especially over a slow connection, and simple tasks like checking that all required fields are filled out or making sure textfields that should contain numbers don't have strings should be performed on the client if at all possible. The `document.forms` property contains an array of `Form` entries contained in the document. As usual in JavaScript, named entries can be accessed by name instead of by number; moreover, named forms are automatically inserted as properties in the `document` object, so any of the following formats would be legal to access forms.

```
var firstForm = document.forms[0];
// Assumes <FORM id="orders" ...>
var orderForm = document.forms["orders"];
// Assumes <FORM id="register" ...>
var registrationForm = document.register;
```

The `Form` object contains an `elements` property that holds an array of `Element` objects. You can retrieve form `elements` by number, by name from the array, or by the property name.

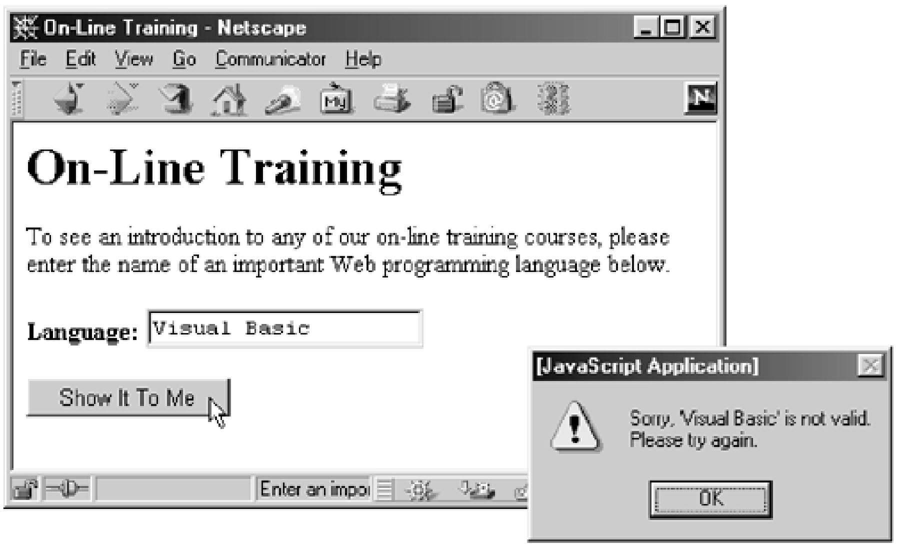
```
var firstElement = firstForm.elements[0];
// Assumes <INPUT ... id="quantity">
var quantityField = orderForm.elements["quantity"];
// Assumes <INPUT ... id="submitSchedule">
var submitButton = register.submitSchedule;
```

Different elements can be manipulated in different ways. Some generally important capabilities include the ability to execute code before a form is submitted (through the `onSubmit` attribute of `FORM`), look up and change form values (through the element's `value` property), to recognize when keyboard focus is gained or lost (through `onFocus` and `onBlur`), and to notice changed values automatically (through `onChange`). The following examples illustrate two major ways form entries are checked: individually (each time one changes) and en masse (only when the form is submitted). For more details, see the `Element` and `Form` objects (Sections 25.6 and 25.8).

Checking Values Individually

Listing 24.13 gives a simple input form containing a single textfield and a `SUBMIT` button. JavaScript is used in two ways. First, when the textfield gets the input focus, text describing the expected value is printed in the status line. The status line is reset when the textfield loses the focus. Second, if the user changes the textfield value to something illegal, a warning is issued when that user leaves the textfield. Then, the textfield value is reset (changing the value through JavaScript does *not* trigger `onChange`), and it is given the input focus for the user to enter a correction. Figure 24-19 shows the results.

Figure 24-19. If the user enters an illegal value, a warning is printed when the keyboard focus leaves the textfield.


On-Line Training - Netscape
File Edit View Go Communicator Help
Show It To Me
Enter an impo
[JavaScript Application]
OK

On-Line Training

To see an introduction to any of our on-line training courses, please enter the name of an important Web programming language below.

Language: Visual Basic

Sorry, 'Visual Basic' is not valid. Please try again.

Listing 24.13 `CheckText.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>On-Line Training</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

// Print a description of the legal text in the status line.

function describeLanguage() {
  status = "Enter an important Web language";
}

// Clear status line.

function clearStatus() {
  status = "";
}
```

```
// When the user changes and leaves textfield, check
// that a valid choice was entered. If not, alert
// user, clear field, and set focus back there.

function checkLanguage() {
  // or document.forms["langForm"].elements["langField"]
  var field = document.langForm.langField;
  var lang = field.value;
  var prefix = lang.substring(0, 4).toUpperCase();
  if (prefix != "JAVA") {
    alert("Sorry, '" + lang + "' is not valid.\n" +
          "Please try again.");
    field.value = "";  // Erase old value
    field.focus();     // Give keyboard focus
  }
}

// -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="WHITE">
<H1>On-Line Training</H1>
<FORM ACTION="cgi-bin/registerLanguage" id="langForm">
To see an introduction to any of our on-line training
courses, please enter the name of an important Web
programming language below.
<P>
<B>Language:</B>
<INPUT TYPE="TEXT" id="langField"
       onFocus="describeLanguage()"
       onBlur="clearStatus()"
       onChange="checkLanguage()">
<P>
<INPUT TYPE="SUBMIT" VALUE="Show It To Me">
</FORM>

</BODY>
</HTML>
```

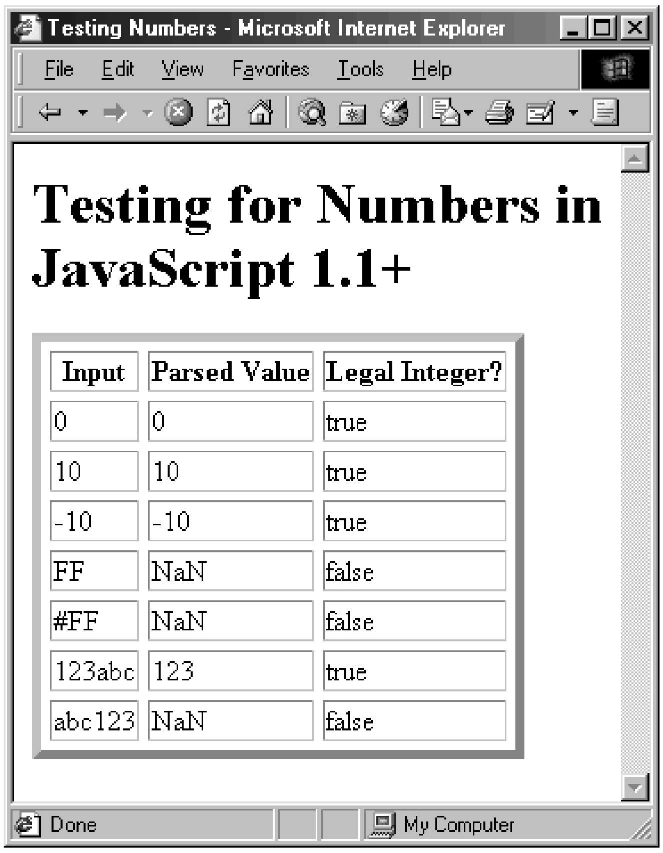
Checking Values When Form Is Submitted

Sometimes it is more convenient to check the entire form in one fell swoop. Some people feel that correcting the user after every mistake is too intrusive, since the user may enter values temporarily but correct them before submission. Other times, it is simply easier to check a bunch of values than to create a separate function for each of several dozen input elements. The key idea is that the function invoked by the `FORM` `onSubmit` attribute prevents submission of the form if the function returns `false`.

Checking numeric values is one of the most common validation tasks, but the `value` property of textfields and text areas is a string, not a number. Fortunately, JavaScript provides two built-in functions to assist in this validation: `parseInt` and `parseFloat`. These functions take a string as input and return either an integer or a floating-point number. In JavaScript 1.2, if no prefix of the string is a valid number, they return the special value `NaN` (not a number), which can be recognized with the built-in `isNaN` function (not by `==`, because NaNs return `false` for all comparisons). Surprisingly, as mentioned in Section 24.1 (Generating HTML Dynamically), JavaScript does not have a property corresponding to "the current language version."

Listing 24.14 illustrates checking input for valid numbers, with the result in Internet Explorer 5.0 shown in Figure 24-20.

Figure 24-20. If the value is not preceded by a number, then the `parseInt` method returns `NaN`.


Testing Numbers - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

Testing for Numbers in JavaScript 1.1+

Input	Parsed Value	Legal Integer?
0	0	true
10	10	true
-10	-10	true
FF	NaN	false
#FF	NaN	false
123abc	123	true
abc123	NaN	false

Listing 24.14 `Numbers.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Testing Numbers</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function isInt(numString) {
    // It's an int if parseInt doesn't return NaN
    return(!isNaN(parseInt(numString)));
}

// -->
</SCRIPT>

</HEAD>
<BODY BGCOLOR="WHITE">
```

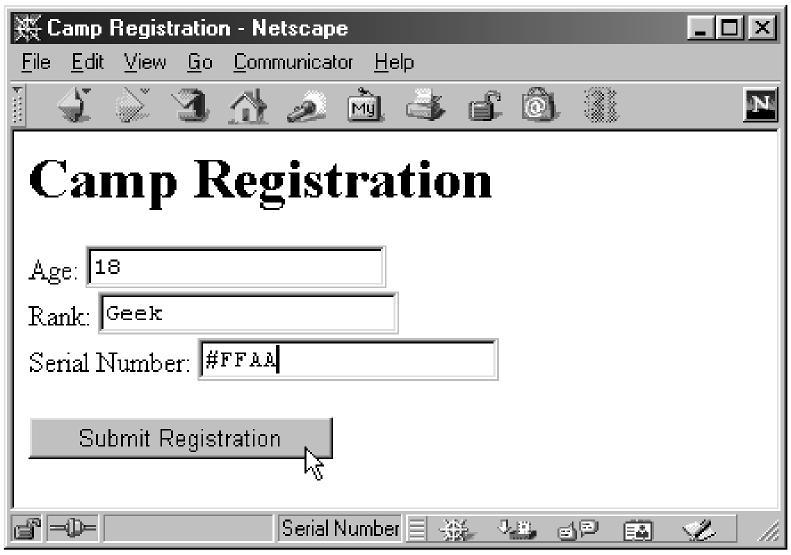
```
<SCRIPT TYPE="text/javascript">
<!--

function testInt(numString) {
  return("<TR><TD>" + numString +
         "<TD>" + parseInt(numString) +
         "<TD>" + isInt(numString) + "\n");
}
document.writeln
  ("<H1>Testing for Numbers in JavaScript 1.2+</H1>\n" +
   "<TABLE BORDER=5 CELLSPACING=5>\n" +
   "<TR><TH>Input<TH>Parsed Value<TH>Legal Integer?\n" +
   testInt("0") +
   testInt("10") +
   testInt("-10") +
   testInt("FF") +
   testInt("#FF") +
   testInt("123abc") +
   testInt("abc123") +
   "</TABLE>");

// -->
</SCRIPT>

</BODY>
</HTML>
```

JavaScript versions earlier than 1.1 do not have an `isNAN` method. However, if a text-field value is supposed to be greater than zero, you can simplify the test considerably by relying on the fact that `NaN` returns `false` when compared to any other number. Here is a variation of `isInt` that uses this idea.

```
function isInt(string) {

  var val = parseInt(string);

  return(val > 0);

}
```

Listing 24.15 uses this approach to create a simple input form with three textfields. The first and third should contain numbers, and the second should contain a string. Rather than correction of values as they are entered, the only action taken during data entry is to print a descriptive message whenever a textfield has the input focus. However, when the form is submitted, the `checkRegistration` function is invoked. This function verifies that the first and third entries are integers and that the second is neither an integer nor missing. Results are shown in Figures 24-21 and 24-22.

Figure 24-21. No checking is done as values are entered.

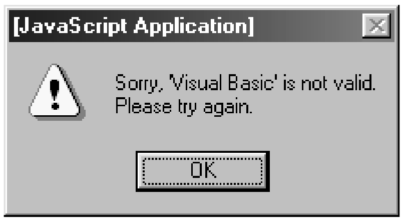

Camp Registration - Netscape
File Edit View Go Communicator Help
Serial Number

Camp Registration

Age: 18
Rank: Geek
Serial Number: #FFAA

Submit Registration

Figure 24-22. Wait! That *is* an integer! Sorry, Charlie, the page designer wasn't planning on hex input.



Sorry, 'Visual Basic' is not valid.
Please try again.

Listing 24.15 `CheckSeveral.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Camp Registration</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function clearStatus() { status = ""; }

function promptAge() { status = "Age (no fractions)"; }

function promptRank() { status = "Rank Name"; }

function promptSerial() { status = "Serial Number"; }

// In JavaScript 1.1+, parseInt returns NaN (recognizable
// through isNaN() for nonintegers. But JavaScript 1.0
// returns 0 and doesn't have an isNaN routine. Since
// comparisons to NaN always fail, the > 0 test works on
// either version.

function isInt(string) {
  var val = parseInt(string);
  return(val > 0);
}
// Four tests:
// 1) Age is an integer.
// 2) Rank is not an integer.
// 3) Rank is not missing.
// 4) Serial number is an integer.
// If any of the tests pass, submission is canceled.

function checkRegistration() {
  var ageField = document.registerForm.ageField;
  if (!isInt(ageField.value)) {
    alert("Age must be an integer.");
    return(false);
  }
  var rankField = document.registerForm.rankField;
  if (isInt(rankField.value)) {
    alert("Use rank name, not rank number.");
    return(false);
  }
  if (rankField.value == "") {
    alert("Missing rank.");
    return(false);
  }
  var serialField = document.registerForm.serialField;
  if (!isInt(serialField.value)) {
    alert("Serial number must be an integer.");
    return(false);
  }
  // Format looks OK. Submit form.
  return(true);
}

// -->
</SCRIPT>

</HEAD>
<BODY BGCOLOR="WHITE">
<H1>Camp Registration</H1>
<FORM ACTION="cgi-bin/register"
      id="registerForm"
      onSubmit="return(checkRegistration())">
Age: <INPUT TYPE="TEXT" id="ageField"
            onFocus="promptAge()"
            onBlur="clearStatus()">
<BR>
Rank: <INPUT TYPE="TEXT" id="rankField"
             onFocus="promptRank()"
             onBlur="clearStatus()">
<BR>
Serial Number: <INPUT TYPE="TEXT" id="serialField"
                      onFocus="promptSerial()"
                      onBlur="clearStatus()">
<P>
```

```
<INPUT TYPE="SUBMIT" VALUE="Submit Registration">
</FORM>

</BODY>
</HTML>
```

Also, be aware that in JavaScript 1.2, Netscape chooses *not* to perform type conversions on operands for the `==` and `!=` operators. For example, consider the following two statements,

```
"123" == 123   // Evaluates to false. JavaScript 1.2 only!
"777" != 777   // Evaluates to true.  JavaScript 1.2 only!
```

In all other versions of JavaScript, Netscape implements type conversion of the operands for these two operators. Internet Explorer performs type conversion for all JScript implementations of ECMAScript and does not suffer from this inconsistency found in JavaScript 1.2.

Core Note

In JavaScript 1.2, Netscape does not perform type conversions on the operands for `==` and `!=`. This behavior is inconsistent with all other versions of JavaScript.

24.7 Using JavaScript to Store and Examine Cookies

A cookie is a small amount of textual information about a page that is stored by the browser on the client system. A cookie can also be manipulated entirely on the client through the use of the `document.cookie` property. This property behaves in a very unusual fashion. If you look up the value of `document.cookie`, you will get a *single* big string containing all the cookie values, as sent by the browser through the `Cookie` HTTP request header. For example, if the current page has three cookies `name1`, `name2`, and `name3`, the value of `document.cookie` would be something like

```
"name1=val1; name2=val2; name3=val3"
```

However, you do not assign values to `document.cookie` by using a single large string like this. Instead, you specify a single cookie at a time, using the same format as would be used in the `Set-Cookie` HTTP response header. See Section 19.11 (Cookies) for a complete description of the syntax, but here are a couple of examples.

```
document.cookie = "name1=val1";
document.cookie = "name2=val2; expires=" + someDate;
document.cookie = "name3=val3; path=/; domain=test.com";
```

Each time `document.cookie` is set, the cookie is stored by the browser. The cookies persist as long as the browser remains open, and if an expiration date is specified, the cookie is reloaded in a later session. This works in any version of Netscape, but it only works in Internet Explorer 5.0 and later.

Core Warning

Cookies are ignored by Internet Explorer 4.x and earlier when the page comes from a local file.

To illustrate the use of cookies, Listing 24.16 creates a simplified order form for "Widgets R Us" corporation. When the form is submitted, the first and last name and account number are stored in cookies. A user can store the name and number without submitting an order by clicking the Register

Account button. When the page is visited later in the same or a different session, the cookie values are used to fill in the first three textfields automatically as a time-saving feature for the user.

There are two particular things you should pay attention to in this example. The first is that the function that fills the textfield values is invoked from the `onLoad` attribute of `BODY`. This guarantees that the page is done loading before the function is invoked, so there is no risk of trying to access textfields that do not yet exist. The second thing to note is the `cookieVal` function, which takes a cookie name (e.g., `"name2"`) and a cookie string (e.g., `"name1=val1; name2=val2; name3=val3"`) and returns the value associated with the name (e.g., `"val2"`).

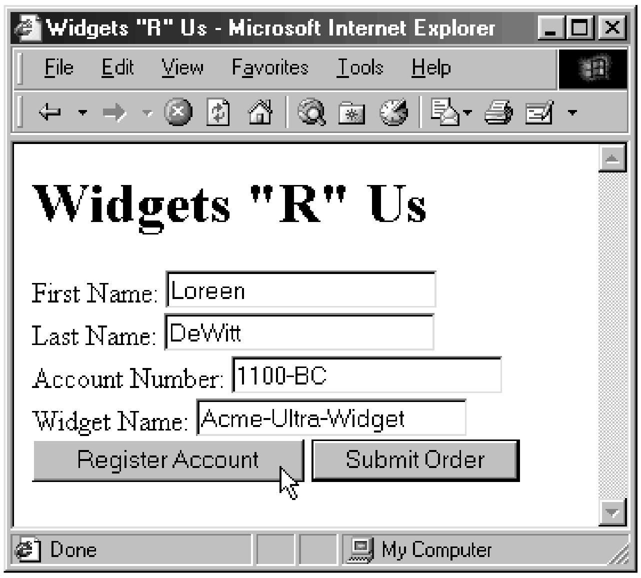
```
function cookieVal(cookieName, cookieString) {
  var startLoc = cookieString.indexOf(cookieName);
  if (startLoc == -1) {
    return("");  // No such cookie
  }
  var sepLoc = cookieString.indexOf("=", startLoc);
  var endLoc = cookieString.indexOf(";", startLoc);
  if (endLoc == -1) { // Last one has no ";"
    endLoc = cookieString.length;
  }
  return(cookieString.substring(sepLoc+1, endLoc));
}
```

This function uses two important methods of the `String` class: `indexOf` and `substring`. The first of these takes two strings as arguments and, if the first appears in the second, returns the location in the second string corresponding to the leftmost occurrence of the first string. There is also a `lastIndexOf` method that returns the starting index of the rightmost occurrence. In either case, `-1` is returned if the second string does not contain the first. The `substring` method takes a start index and an end index and returns the string starting at the start index (inclusive) and going up to the end index (exclusive). The `cookieVal` function might have been a little simpler if the `split` method had been used; this method takes a string and returns an array derived from breaking the string at white space or at a user-specified delimiter. See Section 25.31 for a complete description of the `String` object.

This version of `cookieVal` does not distinguish between an empty value (e.g., the value corresponding to `"bar"` in `"foo=a; bar=; baz=c"`) and a missing cookie (e.g., the value corresponding to `"quux"` in `"foo=a; bar=; baz=c"`). If you want to differentiate these two cases, simply return `null` instead of the empty string from the line containing the "No such cookie" comment. In this case, however, returning an empty string either way simplifies the processing considerably.

Figures 24-23 and 24-24 show the before and after results in Internet Explorer 5.0. Cookies can be hard to debug. To see if any cookies are present for the currently loaded page, you can enter

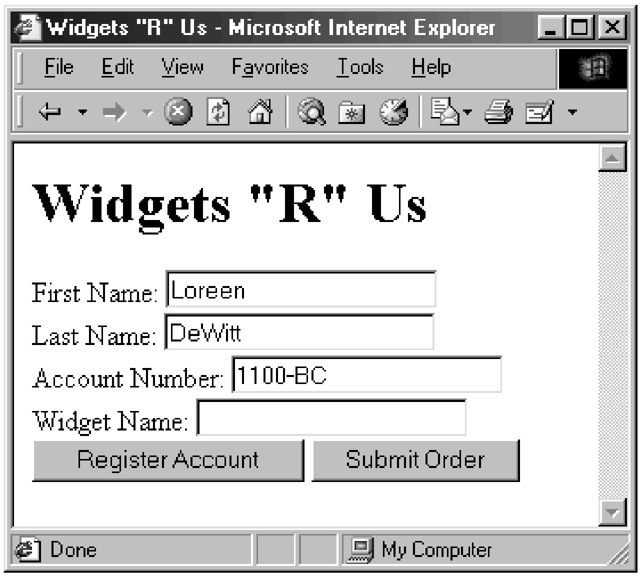
Figure 24-23. The initial page lets the user order widgets. The name and account number are stored when the order is submitted or if Register Account is pressed.


Widgets "R" Us - Microsoft Internet Explorer

Widgets "R" Us

First Name: Loreen
Last Name: DeWitt
Account Number: 1100-BC
Widget Name: Acme-Ultra-Widget

Figure 24-24. Even after Loreen quits and restarts the browser at a later time, the previous name and account number are automatically filled in.


Widgets "R" Us - Microsoft Internet Explorer

Widgets "R" Us

First Name: Loreen
Last Name: DeWitt
Account Number: 1100-BC
Widget Name:

```
javascript:alert(document.cookie)
```

in the browser URL.

Listing 24.16 `Widgets.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Widgets "R" Us</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

// Read last name, first name, and account number
// from textfields and store as cookies.

function storeCookies() {
  var expires = "; expires=Monday, 01-Dec-01 23:59:59 GMT";
  var first = document.widgetForm.firstField.value;
  var last = document.widgetForm.lastField.value;
  var account = document.widgetForm.accountField.value;
  document.cookie = "first=" + first + expires;
  document.cookie = "last=" + last + expires;
  document.cookie = "account=" + account + expires;
}

// Store cookies and give user confirmation.

function registerAccount() {
  storeCookies();
  alert("Registration Successful.");
}
// This does not distinguish an empty cookie from no cookie
// at all, since it doesn't matter here.

function cookieVal(cookieName, cookieString) {
  var startLoc = cookieString.indexOf(cookieName);
  if (startLoc == -1) {
    return("");  // No such cookie
  }
  var sepLoc = cookieString.indexOf("=", startLoc);
  var endLoc = cookieString.indexOf(";", startLoc);
  if (endLoc == -1) { // Last one has no ";"
    endLoc = cookieString.length;
  }
  return(cookieString.substring(sepLoc+1, endLoc));
}

// If cookie values for name or account exist,
// fill in textfields with them.

function presetValues() {
  var firstField = document.widgetForm.firstField;
  var lastField = document.widgetForm.lastField;
  var accountField = document.widgetForm.accountField;
  var cookies = document.cookie;
  firstField.value = cookieVal("first", cookies);
  lastField.value = cookieVal("last", cookies);
  accountField.value = cookieVal("account", cookies);
}

// -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="WHITE" onLoad="presetValues()">

<H1>Widgets "R" Us</H1>

<FORM ACTION="servlet/cwp.Widgets"
      id="widgetForm"
      onSubmit="storeCookies()">
First Name: <INPUT TYPE="TEXT" id="firstField">
<BR>
Last Name: <INPUT TYPE="TEXT" id="lastField">
<BR>
Account Number: <INPUT TYPE="TEXT" id="accountField">
<BR>
Widget Name: <INPUT TYPE="TEXT" id="widgetField">
<BR>
<INPUT TYPE="BUTTON" VALUE="Register Account"
       onClick="registerAccount()">
<INPUT TYPE="SUBMIT" VALUE="Submit Order">

</FORM>
</BODY>
</HTML>
```

24.8 Using JavaScript to Interact with Frames

JavaScript gives you a variety of tools for accessing and manipulating frames. The default `Window` object contains a `frames` property holding an array of frames (other `Window` objects) contained by the current window or frame. It also has `parent` and `top` properties referring to the directly enclosing frame or window and the top-level window, respectively. All of the properties of `Window` can be applied to any of these entries; see Section 25.35 (The Window Object) for details on the available properties. We show two examples here: having one frame direct another to display a particular URL and transferring the input focus to a frame when its contents change. Two additional examples were given in Chapter 4 (Frames). The first showed how you could update the contents of two frames at once, and the second showed how to use JavaScript to prevent your document from appearing as a frame in someone else's Web page. For details, see Section 4.5 (Solving Common Frame Problems).

Directing a Particular Frame to Display a URL

Because each of the frames specified by `frames`, `top`, and `bottom` has a `location` property that, when changed, redirects the window to a new URL, it is quite straightforward for JavaScript code in one frame to force other frames to show a particular URL. Once you have a reference to the parent frame of the one you want to change, you simply look up the frame in the `frames` array and set `location`. Here is an example.

```
someFrame.frames["frameName"].location = "url";
```

Alternatively, because the HTML Document Object Model automatically creates a property for each frame in a window, you could do:

```
someFrame.frames.frameName.location = "url";
```

The key point to changing the content in a frame is that you have to know enough about the frame structure to follow the `parent`, `top`, and `frames` links to the frame of interest. Alternatively, you can create new windows at run time and direct documents to be displayed there. Section 25.35 (The Window Object) gives several examples of this process.

Listing 24.17 creates a page that initially displays two frames: `GetURL.html` (Listing 24.18) and `DisplayURL.html` (Listing 24.19). The top frame contains a textfield for collecting a URL of interest. Entering a value and pressing the Show URL button instructs the other frame to display the specified URL. The process is amazingly simple. First, create a function to display the URL, as follows:

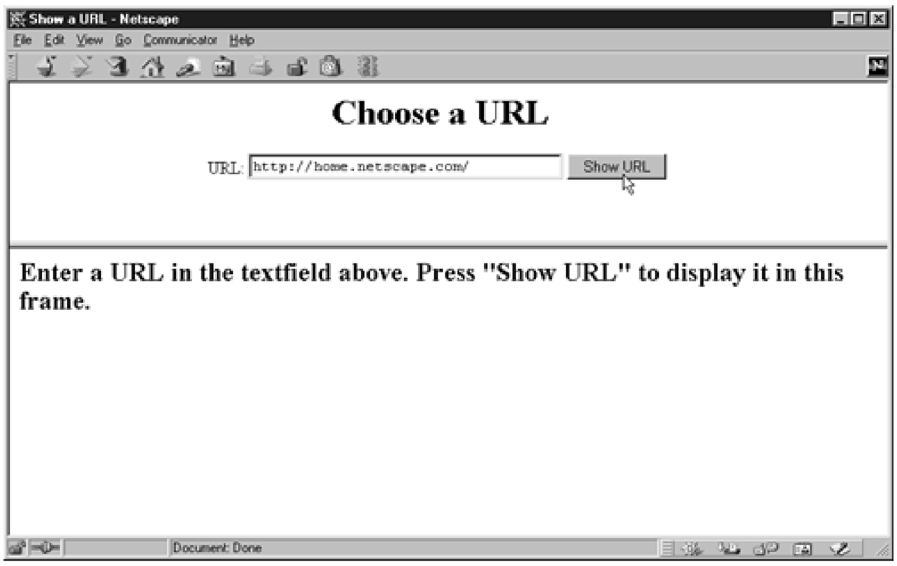
```
function showURL() {
  var url = document.urlForm.urlField.value;
  parent.displayFrame.location = url;
}
```

Next, attach this function to a button by using the `onClick` attribute, as in the following snippet.

```
<INPUT TYPE="BUTTON" VALUE="Show URL"
       onClick="showURL()">
```

That's all there is to it. Figures 24-25 and 24-26 show the results in Internet Explorer.

Figure 24-25. The textfield's initial value is set for Netscape, but arbitrary values can be entered.

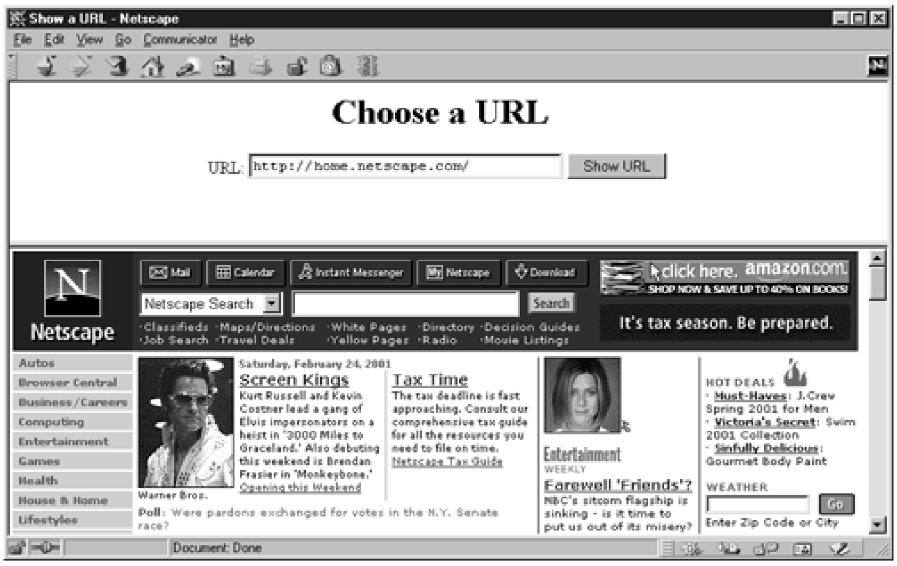


Choose a URL

URL: http://home.netscape.com/

Enter a URL in the textfield above. Press "Show URL" to display it in this frame.

Figure 24-26. Pressing Show URL instructs the bottom frame to show the URL entered in the textfield of the top frame. [© 2001 Netscape Communications Corp. Used with permission. All rights reserved.]



Choose a URL

URL: http://home.netscape.com/

Saturday, February 24, 2001

Screen Kings
Kurt Russell and Kevin Costner lead a gang of Elvis impersonators on a heist in '3000 Miles to Graceland.' Also debuting this weekend is Brendan Frasier in 'Monkeybone.' Opening this Weekend

Warner Bros.

Tax Time
The tax deadline is fast approaching. Consult our comprehensive tax guide for all the resources you need to file on time. Netscape Tax Guide

Poll: Were pardons exchanged for votes in the N.Y. Senate race?

Farewell 'Friends'?
NBC's sitcom flagship is sinking - is it time to put us out of its misery?

HOT DEALS
· Must-Haves: J.Crew Spring 2001 for Men
· Victoria's Secret: Swim 2001 Collection
· Sinfully Delicious: Gourmet Body Paint

Enter Zip Code or City

Listing 24.17 `ShowURL.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
```

```
  <TITLE>Show a URL</TITLE>
</HEAD>

<FRAMESET ROWS="150, *">
  <FRAME SRC="GetURL.html" id="inputFrame">
  <FRAME SRC="DisplayURL.html" id="displayFrame">
</FRAMESET>

</HTML>
```

Listing 24.18 `GetURL.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Choose a URL</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function showURL() {
  var url = document.urlForm.urlField.value;
  // or parent.frames["displayFrame"].location = url;
  parent.displayFrame.location = url;
}

function preloadUrl() {
  if (navigator.appName == "Netscape") {
    document.urlForm.urlField.value =
      "http://home.netscape.com/";
  } else {
    document.urlForm.urlField.value =
      "http://www.microsoft.com/";
  }
}

// -->
</SCRIPT>
</HEAD>

<BODY BGCOLOR="WHITE" onLoad="preloadUrl()">
<H1 ALIGN="CENTER">Choose a URL</H1>

<CENTER>
<FORM id="urlForm">
URL: <INPUT TYPE="TEXT" id="urlField" SIZE=35>
<INPUT TYPE="BUTTON" VALUE="Show URL"
       onClick="showURL()">
</FORM>
</CENTER>

</BODY>
</HTML>
```

Listing 24.19 `DisplayURL.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Display URL</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">
<H2>Enter a URL in the textfield above. Press
"Show URL" to display it in this frame.</H2>

</BODY>
</HTML>
```

Giving a Frame the Input Focus

One of the surprising features of frames is that when you click on a hypertext link that directs the result to a particular frame, that frame does not automatically get the input focus. This can be annoying for a couple of reasons. First, it can give unintuitive results when printed. Suppose that a page is using a small borderless frame to act as a navigation bar. Now, you click in that toolbar to display a page of interest, then select Print. What do you get? A nice printout of the toolbar, that's what! Hardly what you were expecting. Second, it can decrease the value of keyboard shortcuts. Some users commonly use keyboard shortcuts to scroll in large documents. For instance, the up and down arrows cause scrolling on most systems, but only if the window has the input focus. So, if you use a hypertext link that sends a document to a frame, you have to click in that frame before using the keyboard shortcuts.

When JavaScript code is being used to display frames, the fix is trivial: just include a call to `focus`. For instance, here is an improved version of `showURL` that can be used in `GetURL.html` to give the bottom frame the input focus when a URL is sent there.

```
function showURL() {
  var url = document.urlForm.urlField.value;
  parent.displayFrame.location = url;
  // Give frame the input focus
  parent.displayFrame.focus();
}
```

Fixing the problem in regular HTML documents is a bit more tedious. It requires adding `onClick` handlers that call `focus` to each and every occurrence of `A` and `AREA` that includes a `TARGET`, and a similar `onSubmit` handler to each `FORM` that uses `TARGET`.

24.9 Accessing Java from JavaScript

Netscape 3.0 introduced a package called LiveConnect that allows JavaScript to talk to Java, and vice versa. This capability is very important because prior to the introduction of LiveConnect, applets were self-contained programs that had little knowledge of or interaction with the rest of the Web page that contained them. Now, applets can manipulate frames and windows, control images loaded from HTML, read and set values of form elements, read and store cookies, and all the other things that were previously restricted to JavaScript. This section discusses ways to use Java routines and control applets from within JavaScript. Section 24.10 explains the other side of the process: using JavaScript capabilities from within an applet. Here, we discuss three classes of applications.

- *Calling Java methods directly.* In particular, this section shows how to print debugging messages to the Java console.

- *Using applets to perform operations for JavaScript.* In particular, this section shows how a hidden applet can obtain the client hostname, information not otherwise available to JavaScript.

- *Controlling applets from JavaScript.* In particular, this section shows how LiveConnect allows user actions in the HTML part of the page to trigger actions in the applet.

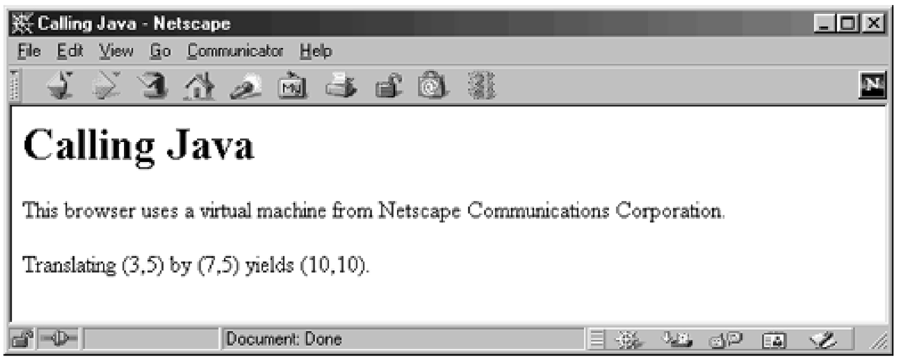
Calling Java Methods Directly

JavaScript can access Java variables and methods simply by using the fully qualified name. For instance, using

```
java.lang.System.out.println("Hello Console");
```

will send the string `"Hello Console"` to the Java console. This is a useful debugging tool when you are developing JavaScript-enabled Web pages. You can also use `new` from within JavaScript to construct Java classes. For example, Listing 24.20 creates a simple page, using the `getProperty` method `java.lang.System` and an instance of the `java.awt.Point` class. Figure 24-27 shows the result.

Figure 24-27. JavaScript can utilize Java even if the current page does not contain an applet.



Calling Java

This browser uses a virtual machine from Netscape Communications Corporation.

Translating (3,5) by (7,5) yields (10,10).

Listing 24.20 `CallJava.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
  <TITLE>Calling Java</TITLE>
</HEAD>
<BODY>
<H1>Calling Java</H1>

<SCRIPT TYPE="text/javascript">
<!--

document.writeln
  ("This browser uses a virtual machine from " +
   java.lang.System.getProperty("java.vendor") + ".");
var pt = new java.awt.Point(3, 5);
pt.translate(7, 5);
document.writeln("<P>");
document.writeln("Translating (3,5) by (7,5) yields (" +
                 pt.x + "," + pt.y + ").");

// -->
</SCRIPT>

</BODY>
```

```
</HTML>
```

Use of Java from JavaScript has two major limitations. First, you cannot perform any operation that would not be permitted in an applet, so you cannot use Java to open local files, call local programs, discover the user's login name, or execute any other such restricted operation. Second, and most significantly, JavaScript provides no mechanism for writing Java methods or creating subclasses. As a result, you will want to create an applet for all but the simplest uses of `java.lang.System.out.println` or a data structure like `java.util.StringTokenizer`.

Using Applets to Perform Operations for JavaScript

With its strong object-oriented framework and (on most platforms) Just In Time compiler, Java is better suited to writing complex data structures and performing long computations than is JavaScript. If JavaScript needs such things, a good alternative is to write them in Java, include them in a "hidden" applet, and call them from JavaScript. JavaScript can access applets either through the `document.applets` array or, if the applet is named, through `document.appletName`. Any public method of the applet can be called by JavaScript. For example, suppose that the applet `Acoustics` has a simple model for computing sound propagation through water and you want to create a Web page to demonstrate the model. You could include the applet in your page with

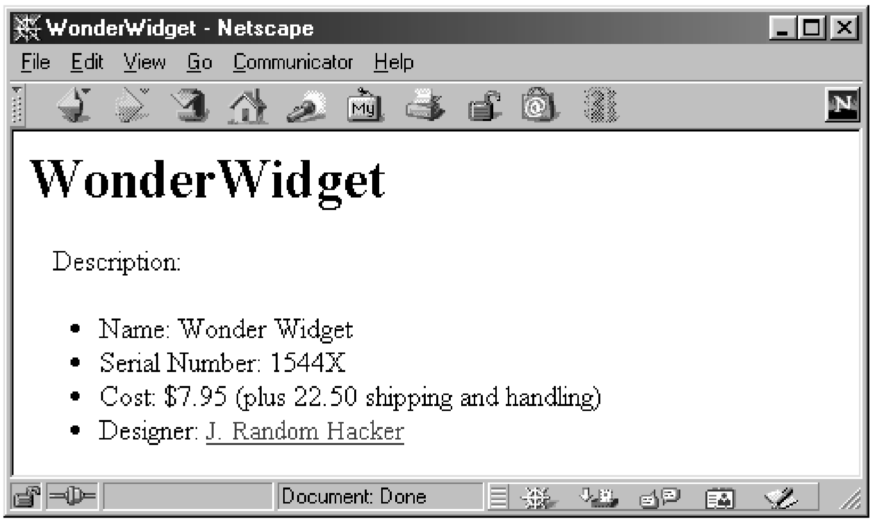
```
<APPLET CODE="Acoustics" WIDTH=10 HEIGHT=10
        id="acoustics">
</APPLET>
```

You could then call the public `getSignalExcess` method as follows:

```
function signalExcess(...) {
  return(document.acoustics.getSignalExcess(...));
}
```

To illustrate, Listing 24.21 creates a Web page with a hypertext link that takes visitors to one of two different resumés, depending on the domain of the client system. Since JavaScript doesn't have a way of determining the client hostname, the page uses a simple hidden applet (Figure 24-28, Listing 24.22) that uses `InetAddress.getLocalHost` to determine this information. If the hostname is inside the author's own company network, the author displays a corporately-politically-correct resumé (Figure 24-29, Listing 24.23), while outside readers get a very different result (Figure 24-30, Listing 24.24) when clicking on the same link.

Figure 24-28. This page contains an applet, even though the applet doesn't contribute to the appearance of the page.

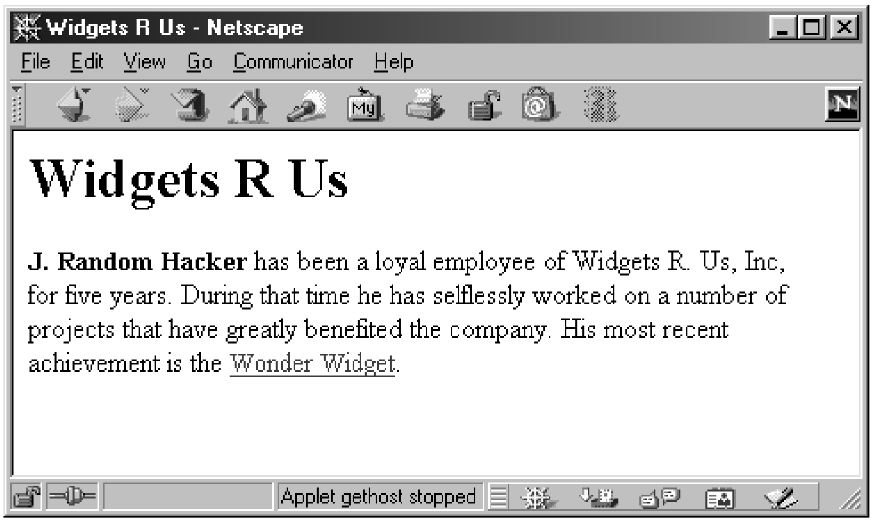


WonderWidget

Description:

- Name: Wonder Widget
- Serial Number: 1544X
- Cost: $7.95 (plus 22.50 shipping and handling)
- Designer: J. Random Hacker

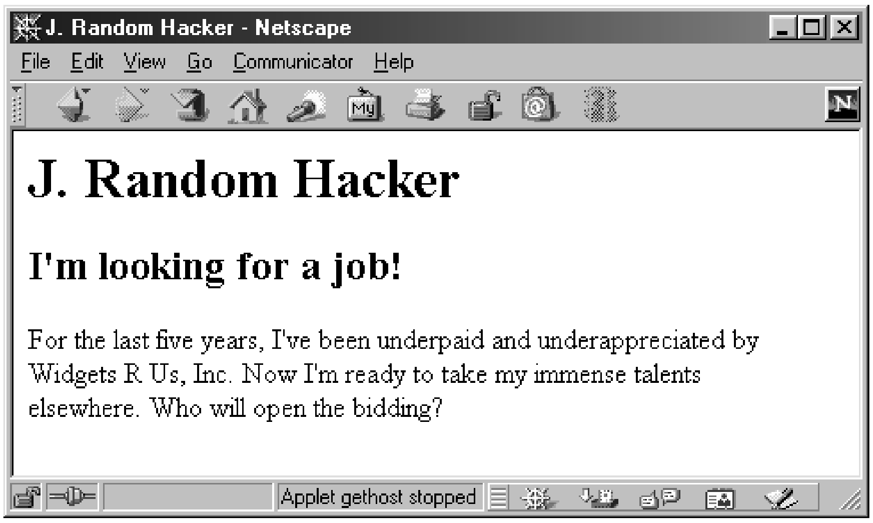
Figure 24-29. If the "J. Random Hacker" link is selected from within the widgets-r-us domain or from a browser that has JavaScript disabled, this first resumé is displayed.



Widgets R Us

J. Random Hacker has been a loyal employee of Widgets R. Us, Inc, for five years. During that time he has selflessly worked on a number of projects that have greatly benefited the company. His most recent achievement is the Wonder Widget.

Figure 24-30. If Java reports a "safe" hostname, a different resumé is displayed.


J. Random Hacker - Netscape
File Edit View Go Communicator Help
Applet gethost stopped

J. Random Hacker

I'm looking for a job!

For the last five years, I've been underpaid and underappreciated by Widgets R Us, Inc. Now I'm ready to take my immense talents elsewhere. Who will open the bidding?

Listing 24.21 `Wonder-Widget.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>WonderWidget</TITLE>

<SCRIPT TYPE="text/javascript">
<!--

function contains(string, substring) {
  return(string.indexOf(substring) != -1);
}

function showResume() {
  if (contains(document.gethost.getHost(),
               "widgets-r-us.com")) {
    location = "ResumeLoyal.html";
  } else {
    location = "ResumeReal.html";
  }
  return(false);
}

// -->
</SCRIPT>

</HEAD>
<BODY BGCOLOR="WHITE">
<H1>WonderWidget</H1>

<APPLET CODE="GetHost" WIDTH=10 HEIGHT=10 id="gethost">
</APPLET>
Description:
<UL>
  <LI>Name: Wonder Widget
  <LI>Serial Number: 1544X
  <LI>Cost: $7.95 (plus 22.50 shipping and handling)
  <LI>Designer:
      <A HREF="ResumeLoyal.html" onClick="return(showResume())">
      J. Random Hacker</A>

</BODY>
</HTML>
```

Listing 24.22 `GetHost.java`

```
import java.applet.Applet;
import java.awt.*;
import java.net.*;

public class GetHost extends Applet {
  private String host;

  public void init() {
    setBackground(Color.white);
    try {
      host = InetAddress.getLocalHost().toString();
    } catch(UnknownHostException uhe) {
      host = "Unknown Host";
    }
  }

  public String getHost() {
    return(host);
  }
}
```

Listing 24.23 `ResumeLoyal.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitioanl//EN">
<HTML>
<HEAD>
```

```
  <TITLE>Widgets R Us</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">
<H1>Widgets R Us</H1>
<B>J. Random Hacker</B> has been a loyal employee of Widgets
R. Us, Inc, for five years. During that time he has
selflessly worked on a number of projects that have greatly
benefited the company. His most recent achievement is the
<A HREF="Wonder-Widget.html">Wonder Widget</A>.
</BODY>
</HTML>
```

Listing 24.24 **`ResumeReal.html`**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>J. Random Hacker</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">
<H1>J. Random Hacker</H1>
<H2>I'm looking for a job!</H2>
For the last five years, I've been underpaid and
underappreciated by Widgets R Us, Inc. Now I'm ready to take
my immense talents elsewhere. Who will open the bidding?
</BODY>
</HTML>
```

Controlling Applets from JavaScript

If an applet is designed with public methods to start it, stop it, customize its appearance, and control its behavior, the applet can be completely controlled from JavaScript. There are several reasons why you might want to do this in JavaScript instead of using GUI controls in Java. First, you might want to mix the controls with formatted text, something HTML excels at. For example, you might want to put controls into a table, and it is far easier to make nicely formatted tables in HTML than in Java. Second, you might want the applet actions to correspond to user events such as submitting a form, something Java could not detect by itself. Third, embedding controls in JavaScript lets you perform a consistent set of actions for *all* applets in a page. For example, Listing 24.25 gives a Web page that lets you control any number of "simulations" showing mold growth. You can insert any number of applets; the Start and Stop buttons apply to every applet on the page. The results are shown in Figures 24-31 and 24-32.

Figure 24-31. Initially, none of the simulations are running.

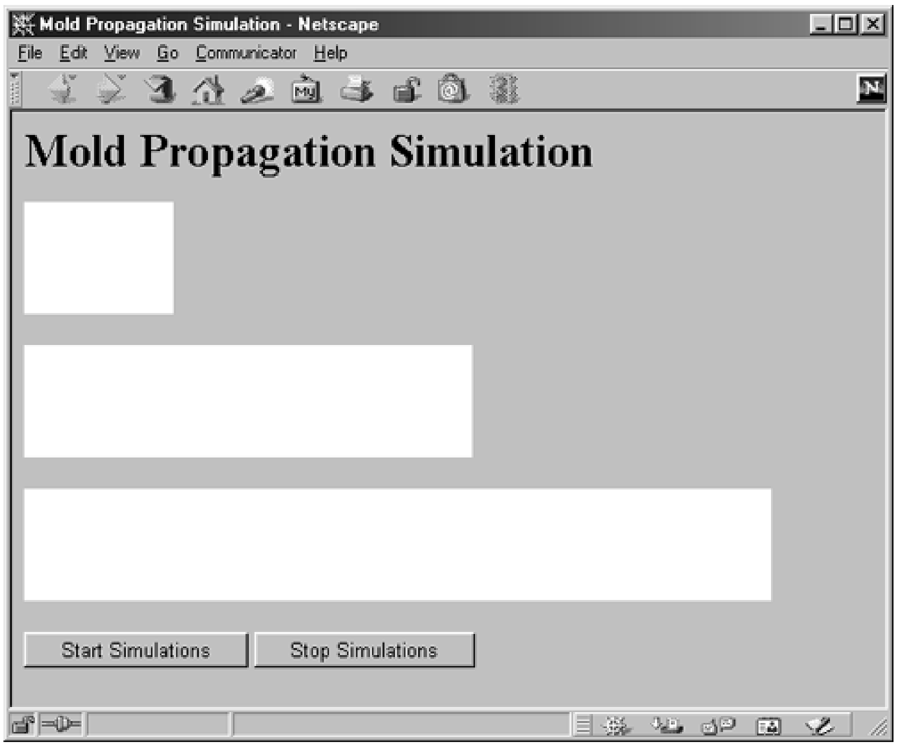
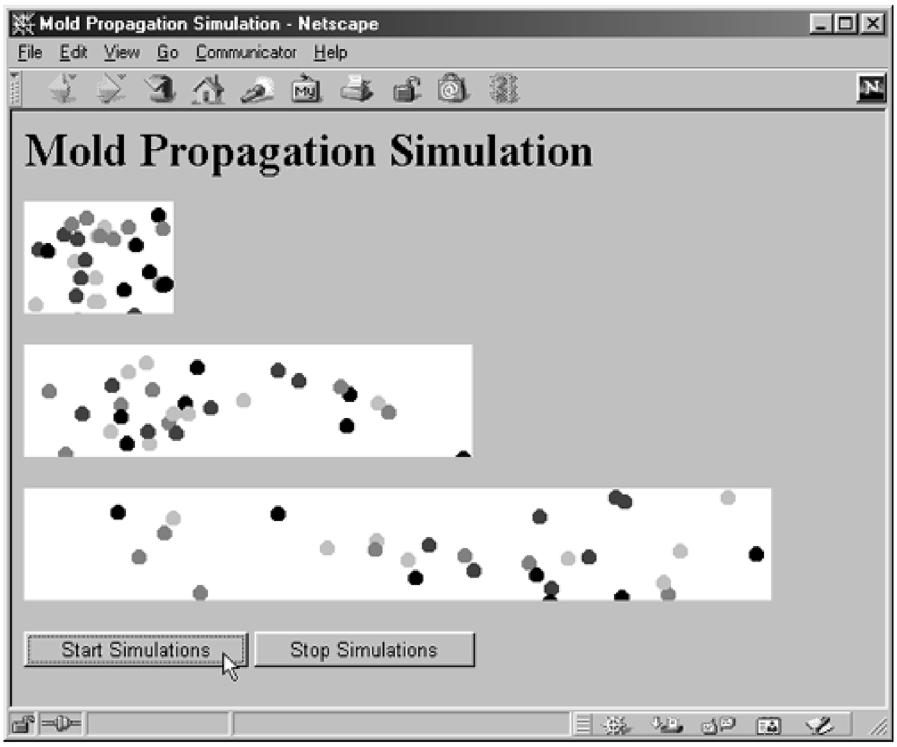

Mold Propagation Simulation - Netscape
File Edit View Go Communicator Help
Mold Propagation Simulation
Start Simulations
Stop Simulations

Figure 24-32. All three applet simulations can be controlled from a single HTML button. Adding additional applets does not require any changes to the Java or JavaScript code.


Mold Propagation Simulation - Netscape
File Edit View Go Communicator Help
Mold Propagation Simulation
Start Simulations
Stop Simulations

Listing 24.25 `MoldSimulation.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Mold Propagation Simulation</TITLE>

<SCRIPT TYPE="text/javascript">
```

```
<!--

// Start simulation for all applets in document.

function startCircles() {
  for(var i=0; i<document.applets.length; i++) {
    document.applets[i].startCircles();
  }
}

// Stop simulation for all applets in document.

function stopCircles() {
  for(var i=0; i<document.applets.length; i++) {
    document.applets[i].stopCircles();
  }
}

// -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#C0C0C0">
<H1>Mold Propagation Simulation</H1>

<APPLET CODE="RandomCircles.class" WIDTH=100 HEIGHT=75>
</APPLET>
<P>
<APPLET CODE="RandomCircles.class" WIDTH=300 HEIGHT=75>
</APPLET>
<P>
<APPLET CODE="RandomCircles.class" WIDTH=500 HEIGHT=75>
</APPLET>
<FORM>
<INPUT TYPE="BUTTON" VALUE="Start Simulations"
       onClick="startCircles()">
<INPUT TYPE="BUTTON" VALUE="Stop Simulations"
       onClick="stopCircles()">
</FORM>

</BODY>
</HTML>
```

Listing 24.26 `RandomCircles.java`

```
import java.applet.Applet;
import java.awt.*;

/** Draw random circles in a background thread.
 *  Needs an external event to start/stop drawing.
 */

public class RandomCircles extends Applet
                           implements Runnable {
  private boolean drawCircles = false;
  public void init() {
    setBackground(Color.white);
  }

  public void startCircles() {
    Thread t = new Thread(this);
    t.start();
  }

  public void run() {
    Color[] colors = { Color.lightGray, Color.gray,
                       Color.darkGray, Color.black };
    int colorIndex = 0;
    int x, y;
    int width = getSize().width;
    int height = getSize().height;

    Graphics g = getGraphics();
    drawCircles = true;
    while(drawCircles) {
      x = (int)Math.round(width * Math.random());
      y = (int)Math.round(height * Math.random());
      g.setColor(colors[colorIndex]);
      colorIndex = (colorIndex + 1) % colors.length;
      g.fillOval(x, y, 10, 10);
      pause(0.1);
    }
  }

  public void stopCircles() {
    drawCircles = false;
  }

  private void pause(double seconds) {
    try {
      Thread.sleep((int)(Math.round(seconds * 1000.0)));
    } catch(InterruptedException ie) {}
  }
}
```

24.10 Accessing JavaScript from Java

Not only does LiveConnect let you call Java methods and control applets from JavaScript, but it also lets applets access JavaScript. The `netscape.javascript.JSObject` class lets you use Java syntax to access all JavaScript objects, read and set all available properties, and call any legal method. Furthermore, you can use the `eval` method to invoke arbitrary JavaScript code when doing so is easier than using Java syntax. This process involves seven steps, as follows:

1. Obtain and install the `JSObject` class.

2. Import the class in your applet.

3. From the applet, obtain a JavaScript reference to the current window.

4. Read the JavaScript properties of interest.
5. Set the JavaScript properties of interest.
6. Call the JavaScript methods of interest.
7. Give the applet permission to access its Web page.

Following are some details on these steps.

Obtain and install the JSObject class The `JSObject` class is provided by Netscape 4, compressed in JAR file titled `java40`. The location of this file will vary among operating systems. On Windows 98, it should be in

```
NetscapeInstallPath\Program\Java\Classes\
```

Use Find from your Start menu if you have trouble finding the file. On Unix, this file will generally be in the Netscape installation directory. If you cannot find it, try the following:

```
Unix> cd /usr/local
Unix> find . -name java40 -print
```

Once you have this file, simply add it to the end of your `CLASSPATH`; the Java compiler knows how to look inside JAR files already. If you do this, be sure to unset `CLASSPATH` before trying to use appletviewer; it will not function properly otherwise. In fact, since both Netscape and Internet Explorer grant special privileges to classes that appear in the `CLASSPATH`, it is best to unset the `CLASSPATH` before starting any browser. If you prefer, you can decompress the JAR file from the command line, `jar xf java40.jar`, extract the `JSObject.class` file, and install it separately. You will *need* to do this if you develop and compile your code on a system that does not have Netscape installed. In addition, you should probably include the `JSObject` class file with your applets, since the client may be using a different browser vendor and version.

Import JSObject in your applet At the top of your applet code, add the line:

```
import netscape.javascript.JSObject;
```

From the applet, obtain a JavaScript reference to the current window Use the static `getWindow` method of `JSObject` to obtain a reference to the window containing the applet.

```
JSObject window =
  JSObject.getWindow(this); // this=applet
```

A complete list of the available methods in the `JSObject` class is given at the end of this section.

Read the JavaScript properties of interest Use the `getMember` method to read properties of the main JavaScript window. Then, use `getMember` on the results to access properties of other JavaScript objects. For example,

```
JSObject document =
  (JSObject)window.getMember("document");
String cookies =
  (String)document.cookie;
JSObject someForm =
  (JSObject)document.getMember("someFormName");
```

```
JSObject someElement =
  (JSObject)someForm.getMember("someElementName");
```

You can also use `getSlot` with an index to access elements of an array.

Set the JavaScript properties of interest Use the `setMember` method for this step. For example,

```
document.setMember("bgColor", "red");
someElement.setMember("value", "textfield value");
```

Note that the second argument to `setMember` must be a Java `Object`, so primitive objects must be converted to their corresponding wrapper type before being passed. Thus, an `int` named `intValue` must be turned into an `Integer` through `new Integer (intValue)` before being assigned to a property expecting an integer. An alternative is to construct a string corresponding to the assignment in JavaScript, then pass that string to `eval` (see below).

Call the JavaScript methods of interest You call JavaScript methods either by using the `call` method or by constructing a JavaScript expression containing the method calls and passing it to `eval`. The `call` method takes the method name and an array of arguments; `eval` takes a single string. For instance,

```
String[] message = { "An alert message" };
window.call("alert", message);
window.eval("alert('An alert message')");
```

Give the applet permission to access its Web page To prevent Web page authors from accidentally using applets that read or modify their pages (perhaps reporting results over the network), the Web page author must explicitly give the applet permission to access the page. Access is granted through the `MAYSCRIPT` attribute of the `APPLET` element. For example,

```
<APPLET CODE=... WIDTH=... HEIGHT=... MAYSCRIPT>
  ...
</APPLET>
```

Example: Matching Applet Background with Web Page

One of the problems with writing general-purpose applets is that they don't have access to the background color of the Web page that contains them, so they can't adapt their color to match. A common solution is to supply the background color as a `PARAM` entry, but this has the drawback that the color has to be repeated (once in the `BODY` tag, and again in the `PARAM`), risking inconsistency if one is updated without the other. With LiveConnect, the applet can read the background color from the `Document` object and set its color automatically. Listings 24.27 and 24.28 give an example.

Listing 24.27 `MatchColor.java`

```
import java.applet.Applet;
import java.awt.*;
import netscape.javascript.JSObject;

public class MatchColor extends Applet {
  public void init() {
    JSObject window = JSObject.getWindow(this); // this=applet
    JSObject document = (JSObject)window.getMember("document");
    // E.g., "#ff0000" for red
    String pageColor = (String)document.getMember("bgColor");
    // E.g., parseInt("ff0000", 16) --> 16711680
    int bgColor =
          Integer.parseInt(pageColor.substring(1, 7), 16);
    setBackground(new Color(bgColor));
  }
}
```

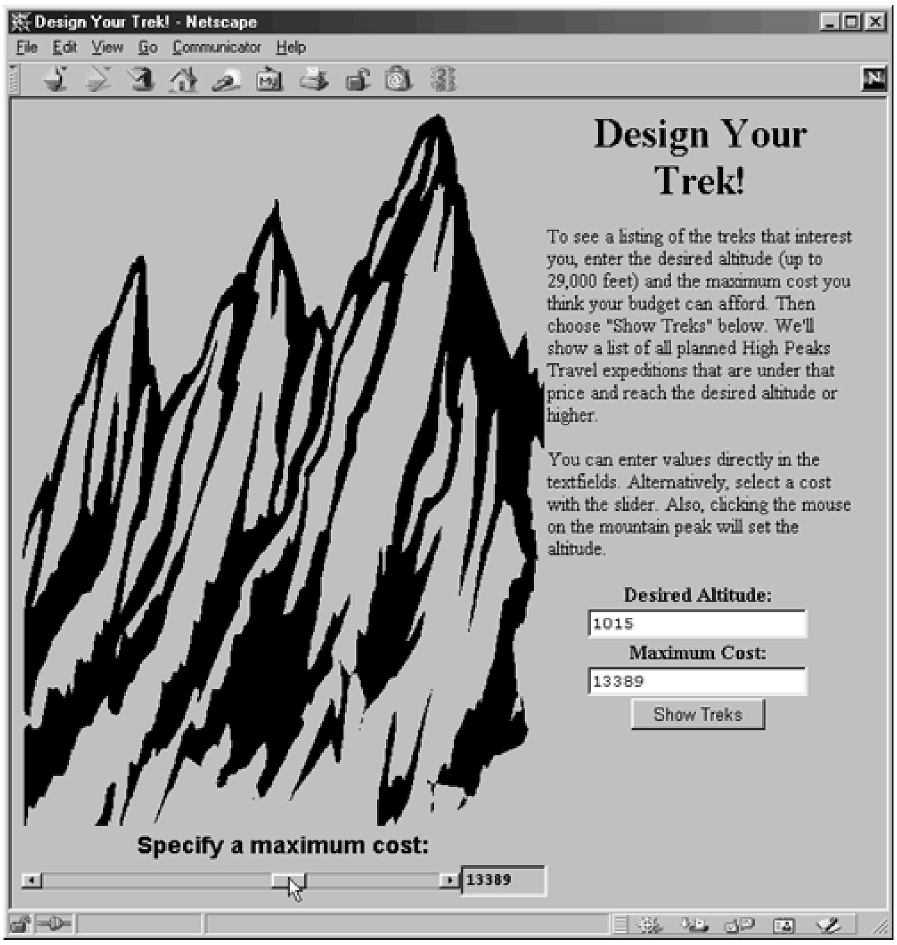
Listing 24.28 `MatchColor.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>MatchColor</TITLE>
</HEAD>
<BODY BGCOLOR="RED">
<H1>MatchColor</H1>
<APPLET CODE="MatchColor.class" WIDTH=300 HEIGHT=300 MAYSCRIPT>
</APPLET>
</BODY>
</HTML>
```

Example: An Applet That Controls HTML Form Values

For a more extensive example, Listing 24.29 creates a page that lets users select mountain climbing trips based upon two criteria: the altitude they want to attain and the budget they want to stay within. On any platform, users can enter these two values directly in the HTML form and submit them to get information on available options. Moving the cost slider changes the value displayed in the HTML form; moving the mouse up and down the mountain changes the value displayed in the altitude form. Figure 24-33 shows the result after the user drags the slider and moves the mouse partway up the mountain.

Figure 24-33. Moving the mouse over the image or dragging the slider changes the values in the HTML textfields.


Design Your Trek! - Netscape
File Edit View Go Communicator Help

Design Your Trek!

To see a listing of the treks that interest you, enter the desired altitude (up to 29,000 feet) and the maximum cost you think your budget can afford. Then choose "Show Treks" below. We'll show a list of all planned High Peaks Travel expeditions that are under that price and reach the desired altitude or higher.

You can enter values directly in the textfields. Alternatively, select a cost with the slider. Also, clicking the mouse on the mountain peak will set the altitude.

Desired Altitude: 1015

Maximum Cost: 13389

Show Treks

Specify a maximum cost: 13389

Listing 24.29 `Everest.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Design Your Trek!</TITLE>
</HEAD>
<BODY>

<APPLET CODE="Everest.class" WIDTH=400 HEIGHT=600
        MAYSCRIPT ALIGN="LEFT">
</APPLET>

<H1 ALIGN="CENTER">Design Your Trek!</H1>
To see a listing of the treks that interest you, enter the
desired altitude (up to 29,000 feet) and the maximum cost you
think your budget can afford. Then choose "Show Treks" below.
We'll show a list of all planned High Peaks Travel expeditions
that are under that price and reach the desired altitude or
higher.
<P>
You can enter values directly in the textfields. Alternatively,
select a cost with the slider. Also, clicking the mouse on the
mountain peak will set the altitude.
<CENTER>
<FORM ACTION="servlet/trekOptions"  id="highPeaksForm">
<B>Desired Altitude:</B>
```

```
<INPUT TYPE="TEXT" id="altitudeField">
<BR>
<B>Maximum Cost:</B>
<INPUT TYPE="TEXT" id="costField">
<BR>
<INPUT TYPE="SUBMIT" VALUE="Show Treks">
</FORM>
</CENTER>
</BODY>
</HTML>
```

Listing 24.30 **`Everest.java`**

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import netscape.javascript.JSObject;

/** An applet that draws a mountain image and displays
 *  a slider. Dragging the slider changes a textfield
 *  in the HTML file containing the applet. Moving the
 *  mouse over the image changes another textfield in the
 *  containing HTML file, using an altitude value
 *  where the bottom of the applet corresponds to 0
 *  and the top to 29000 feet. This requires
 *  an HTML file with a form named "highPeaksForm"
 *  containing two textfields: one named "costField"
 *  and one named "altitudeField". It also requires use
 *  of the MAYSCRIPT tag in the <APPLET ...> declaration.
 */

public class Everest extends Applet {
  private Image mountain;
  private JSObject window, document, highPeaksForm,
          costField, altitudeField;
  private int width, height;

  public void init() {
    setBackground(Color.lightGray);
    mountain = getImage(getCodeBase(), "images/peak5.gif");
    width = getSize().width;
    height = getSize().height;
    // Start image loading immediately.
    prepareImage(mountain, width, height, this);
    setLayout(new BorderLayout());
    Font sliderFont = new Font("Helvetica", Font.BOLD, 18);
    LabeledCostSlider costSlider =
      new LabeledCostSlider("Specify a maximum cost:",
                            sliderFont, 2000, 20000, 5000,
                            this);
    add(costSlider, BorderLayout.SOUTH);
    addMouseMotionListener(new MouseMotionAdapter() {
      // When user moves the mouse, scale the y value
      // from 29000 (top) to 0 (bottom) and send it
      // to external textfield through JavaScript.
      public void mouseMoved(MouseEvent event) {
        System.out.println("Mouse Move at : " + event.getY());
        setAltitudeField((height - event.getY()) * 29000 / height);
      }
    });

    // Get references to HTML textfields through JavaScript.
    window = JSObject.getWindow(this); // this=applet
    document = (JSObject)window.getMember("document");
    highPeaksForm =
      (JSObject)document.getMember("highPeaksForm");
    costField =
      (JSObject)highPeaksForm.getMember("costField");
    altitudeField =
      (JSObject)highPeaksForm.getMember("altitudeField");
    setCostField(5000);
    setAltitudeField(15000);
  }

  public void paint(Graphics g) {
    g.drawImage(mountain, 0, 0, width, height, this);
  }

  /** Change textfield through JavaScript. */

  public void setCostField(int val) {
    costField.setMember("value", String.valueOf(val));
  }

  /** Change textfield through JavaScript. */

  private void setAltitudeField(int val) {
    altitudeField.setMember("value", String.valueOf(val));
  }
}
```

The Everest applet makes use of a `LabeledCostSlider` (shown in the bottom-left region of Figure 24-33) to provide a custom slider with a text label above the slider. The `LabeledCostSlider` source is shown in Listing 24.31. The custom slider, `CostSlider` (Listing 24.32), extends `Slider`, Listing 24.33, and updates the value in the cost field whenever the thumbnail of the slider is adjusted. The `Slider` class combines a horizontal `Scrollbar` and a `TextField` (to the right of the scrollbar) grouped together by a `ScrollbarPanel`, Listing 24.34.

Listing 24.31 `LabeledCostSlider.java`

```
import java.awt.*;

/** A CostSlider with a label centered above it. */

public class LabeledCostSlider extends Panel {
  public LabeledCostSlider(String labelString,
                           Font labelFont,
                           int minValue, int maxValue,
                           int initialValue,
                           Everest app) {
    setLayout(new BorderLayout());
    Label label = new Label(labelString, Label.CENTER);
    if (labelFont != null) {
      label.setFont(labelFont);
    }
    add(label, BorderLayout.NORTH);
    CostSlider slider = new CostSlider(minValue,
                                       maxValue,
                                       initialValue,
                                       app);
    add(slider, BorderLayout.CENTER);
  }
}
```

Listing 24.32 `CostSlider.java`

```
/** A Slider that takes an Everest applet as an argument,
 *  calling back to its setCostField when the slider value
 *  changes.
 */

public class CostSlider extends Slider {
  private Everest app;

  public CostSlider(int minValue, int maxValue,
                    int initialValue, Everest app) {
    super(minValue, maxValue, initialValue);
    this.app = app;
  }

  public void doAction(int value) {
    app.setCostField(value);
  }
}
```

Listing 24.33 `Slider.java`

```
import java.awt.*;
import java.awt.event.*;

/** A class that combines a horizontal Scrollbar and a TextField
 *  (to the right of the Scrollbar). The TextField shows the
 *  current scrollbar value, plus, if setEditable(true) is set,
 *  it can be used to change the value as well.
 */

public class Slider extends Panel implements ActionListener,
                                             AdjustmentListener {
  private Scrollbar scrollbar;
  private TextField textfield;
  private ScrollbarPanel scrollbarPanel;
  private int preferredWidth = 250;
```

```
/** Construct a slider with the specified min, max and initial
 *  values. The "bubble" (thumb) size is set to 1/10th the
 *  scrollbar range.
 */

public Slider(int minValue, int maxValue, int initialValue) {
  this(minValue, maxValue, initialValue,
       (maxValue - minValue)/10);
}

/** Construct a slider with the specified min, max,and initial
 *  values, plus the specified "bubble" (thumb) value. This
 *  bubbleSize should be specified in the units that min and
 *  max use, not in pixels. Thus, if min is 20 and max is 320,
 *  then a bubbleSize of 30 is 10% of the visible range.
 */

public Slider(int minValue, int maxValue, int initialValue,
              int bubbleSize) {
  setLayout(new BorderLayout());
  maxValue = maxValue + bubbleSize;
  scrollbar = new Scrollbar(Scrollbar.HORIZONTAL,
                            initialValue, bubbleSize,
                            minValue, maxValue);
  scrollbar.addAdjustmentListener(this);
  scrollbarPanel = new ScrollbarPanel(6);
  scrollbarPanel.add(scrollbar, BorderLayout.CENTER);
  add(scrollbarPanel, BorderLayout.CENTER);
  textfield = new TextField(numDigits(maxValue) + 1);
  textfield.addActionListener(this);
  setFontSize(12);
  textfield.setEditable(false);
  setTextFieldValue();
  add(textfield, BorderLayout.EAST);
}

/** A placeholder to override for action to be taken when
 *  scrollbar changes.
 */

public void doAction(int value) {
}

/** When textfield changes, sets the scrollbar */

public void actionPerformed(ActionEvent event) {
  String value = textfield.getText();
  int oldValue = getValue();
  try {
    setValue(Integer.parseInt(value.trim()));
  } catch(NumberFormatException nfe) {
    setValue(oldValue);
  }
}

/** When scrollbar changes, sets the textfield */

public void adjustmentValueChanged(AdjustmentEvent event) {
  setTextFieldValue();
  doAction(scrollbar.getValue());
}

/** Returns the Scrollbar part of the Slider. */

public Scrollbar getScrollbar() {
  return(scrollbar);
}

/** Returns the TextField part of the Slider */

public TextField getTextField() {
  return(textfield);
}
/** Changes the preferredSize to take a minimum width, since
 *  super-tiny scrollbars are hard to manipulate.
 */

public Dimension getPreferredSize() {
  Dimension d = super.getPreferredSize();
  d.height = textfield.getPreferredSize().height;
  d.width = Math.max(d.width, preferredWidth);
  return(d);
}

/** This just calls preferredSize */

public Dimension getMinimumSize() {
  return(getPreferredSize());
}

/** To keep scrollbars legible, a minimum width is set. This
 *  returns the current value (default is 150).
 */

public int getPreferredWidth() {
  return(preferredWidth);
}

/** To keep scrollbars legible, a minimum width is set. This
 *  sets the current value (default is 150).
 */

public void setPreferredWidth(int preferredWidth) {
  this.preferredWidth = preferredWidth;
}
/** This returns the current scrollbar value */

public int getValue() {
  return(scrollbar.getValue());
}

/** This assigns the scrollbar value. If it is below the
 *  minimum value or above the maximum, the value is set to
 *  the min and max value, respectively.
 */

public void setValue(int value) {
  scrollbar.setValue(value);
  setTextFieldValue();
}

/** Sometimes horizontal scrollbars look odd if they are very
 *  tall. So empty top/bottom margins can be set. This returns
 *  the margin setting. The default is four.
 */

public int getMargins() {
  return(scrollbarPanel.getMargins());
}

/** Sometimes horizontal scrollbars look odd if they are very
 *  tall. So empty top/bottom margins can be set. This sets
 *  the margin setting.
 */

public void setMargins(int margins) {
  scrollbarPanel.setMargins(margins);
}

/** Returns the current textfield string. In most cases this
 *  is just the same as a String version of getValue, except
 *  that there may be padded blank spaces at the left.
 */

public String getText() {
  return(textfield.getText());
}

/** This sets the TextField value directly. Use with extreme
 *  caution since it does not right-align or check whether
 *  value is numeric.
 */

public void setText(String text) {
  textfield.setText(text);
}
/** Returns the Font being used by the textfield.
 *  Courier bold 12 is the default.
 */

public Font getFont() {
  return(textfield.getFont());
}

/** Changes the Font being used by the textfield. */

public void setFont(Font textFieldFont) {
  textfield.setFont(textFieldFont);
}

/** The size of the current font */

public int getFontSize() {
  return(getFont().getSize());
}

/** Rather than setting the whole font, you can just set the
 *  size (Monospaced bold will be used for the family/face).
 */

public void setFontSize(int size) {
  setFont(new Font("Monospaced", Font.BOLD, size));
}

/** Determines if the textfield is editable. If it is, you can
 *  enter a number to change the scrollbar value. In such a
 *  case, entering a value outside the legal range results in
 *  the min or max legal value. A noninteger is ignored.
 */

public boolean isEditable() {
  return(textfield.isEditable());
}

/** Determines if you can enter values directly into the
 *  textfield to change the scrollbar.
 */

public void setEditable(boolean editable) {
  textfield.setEditable(editable);
}

// Sets a right-aligned textfield number.

private void setTextFieldValue() {
  int value = scrollbar.getValue();
  int digits = numDigits(scrollbar.getMaximum());
  String valueString = padString(value, digits);
  textfield.setText(valueString);
}

// Repeated String concatenation is expensive, but this is
// only used to add a small amount of padding, so converting
// to a StringBuffer would not pay off.

private String padString(int value, int digits) {
  String result = String.valueOf(value);
  for(int i=result.length(); i<digits; i++) {
    result = " " + result;
  }
  return(result + " ");
}

// Determines the number of digits in a decimal number.

private static final double LN10 = Math.log(10.0);

private static int numDigits(int num) {
  return(1 + (int)Math.floor(Math.log((double)num)/LN10));
}
}
```

Listing 24.34 **ScrollbarPanel.java**

```
import java.awt.*;

/** A Panel with adjustable top/bottom insets value.
 *  Used to hold a Scrollbar in the Slider class
 */

public class ScrollbarPanel extends Panel {
  private Insets insets;
  public ScrollbarPanel(int margins) {
    setLayout(new BorderLayout());
    setMargins(margins);
  }

  public Insets insets() {
    return(insets);
  }

  public int getMargins() {
    return(insets.top);
  }

  public void setMargins(int margins) {
    this.insets = new Insets(margins, 0, margins, 0);
  }
}
```

Methods in the JSObject Class

The following methods are available as part of `JSObject`. `JSObject` is `final`, so it cannot be subclassed.

public Object call(String methodName, Object[ ] args)

This method lets you call the JavaScript method of the specified name.

public Object eval(String javaScriptCode)

This method lets you evaluate an arbitrary JavaScript expression.

public Object getMember(String propertyName)

This method returns a property value. Cast the result to the appropriate type.

public Object getSlot(int arrayIndex)

This method returns an array value. Cast the result to the appropriate type.

public static JSObject getWindow(Applet applet)

This *static* method retrieves the JavaScript `Window` corresponding to the one holding the applet.

public void removeMember(String propertyName)

This method deletes a property.

public void setMember(String propertyName, Object value)

This method assigns a value to the specified property.

public void setSlot(int arrayIndex, Object value)

This method places a value in the specified location in the array.

24.11 Summary

This chapter introduced JavaScript and gave examples of the main ways in which JavaScript can be applied to do the following:

- Customize Web pages based on the situation
- Make pages more dynamic
- Validate HTML form input
- Manipulate cookies
- Control frames
- Integrate Java and JavaScript

This chapter did not, however, give a complete description of `Window`, `Document`, `Navigator` and other standard classes in JavaScript 1.2. The Quick Reference given in the next chapter provides a description of the constructors, properties, and methods for the JavaScript objects.

CONTENTS