Learn JavaScript

Chuck Easttom

Wordware Publishing, Inc.

Library of Congress Cataloging-in-Publication Data

Easttom, Chuck.
Learn JavaScript / by Chuck Easttom.
p. cm.
ISBN 1-55622-856-2 (pb)
1. Computer graphics. 2. Computer games. I. Title.
T385.F447 2000
794.8'166--dc21 00-047319
CIP

2320 Los Rios Boulevard
Plano, Texas 75074

Printed in the United States of America

ISBN 1-55622-856-2
10 9 8 7 6 5 4 3 2 1
0109

All inquiries for volume purchases of this book should be addressed to Wordware Publishing, Inc., at the above address. Telephone inquiries may be made by calling:

(972) 423-0090

Contents Summary

Contents

Acknowledgments

Writ ing a book is a very dif fi cult task, and it is never the work of a sin gle person. I would like to take the time to thank some of the peo ple who were indis pens able in mak ing this book hap pen:

Jim Hill and Wes Beckwith, the won der ful guys at Wordware Pub lishing who labored through this with me. With out Jim and Wes, this book sim ply could not have hap pened. Their ideas and input helped cre ate this book. Not to men tion that Jim always paid for lunch!

A very spe cial thanks to Nor man Smith who did an out stand ing tech ni cal review of this book. Norm, you saved me from really embar rass ing myself. Your review work is sim ply top notch.

Finally, I need to thank my wife, Misty, and my son, AJ, who both tol er ated me spend ing hours secluded in my den work ing on this pro ject. With out all of these peo ple, this book would never have hap pened.

Introduction

Prerequisites

This book assumes you have abso lutely no prior knowl edge of JavaScript. A basic work ing knowl edge of HTML is pre ferred but not essen tial. The first two chap ters pro vide a basic intro duc tion on HTML and JavaScript. The rest of the book takes you step by step through the pro cess of build ing an actual web site using both HTML and JavaScript. How ever, the main empha sis of this book is teach ing you JavaScript. After Chap ter 1 “HTML Primer,” all HTML is sim ply shown to pro vide con text for the JavaScript, and an expla na tion for the HTML code is not usu ally given. How ever, only stan dard HTML tech niques are used and the code is pre sented in a very orga nized man ner so that any one with a basic knowl edge of HTML can eas ily fol low along.

As each new con cept is intro duced, I will first explain the con cept in a generic con text and then I will show how it applies spe cif i cally to the web site we are build ing. In this way, you will get both the con cept and the appli ca tion.

How to Read This Book

Some com puter pro gram ming books are meant for you to sim ply take what you need from cer tain chap ters. This book is intended for you to read from start to fin ish. If you fol low the exam ples pro vided, when you com plete the book you will have a com pletely func tional web site with lots of HTML and plenty of inter est ing JavaScripts.

Programming Style

As both an aid to the reader and to illus trate good pro gram ming prac tices, I use a uni form style through out this entire text and in all the sam ple code you will find on the com pan ion CD. I also have a ten dency to com ment very

heavily—some would say too much. This comes from teach ing and cre at ing code sam ples for stu dents. I hope you will find this help ful:

```
for (var intcounter = 0; intcounter < 10; ++intcounter)
{
   [JavaScript statements]
}// end of for loop
```

Some JavaScript scripters pre fer to use a dif fer ent style:

```
for (var num = 0; num < 10; ++num){
   [JavaScript statements]
}
```

A lot of my style pref er ences are just that, pref er ences. It is hard to call one style right and another wrong. How ever, I strongly sug gest when you are writing code to con sider the fact that other pro gram mers will prob a bly need to read your code at some point. You don't want them to have abso lutely no idea what you were think ing when you wrote your code. I try to write code in such a way that even a nov ice pro gram mer with no prior knowl edge of the pro ject at hand could eas ily deduce the intended pur pose of the code.

NOTE: Many tags in HTML must be closed. The <B> tag must be closed with </B>. However some tags, such as <TD> and <TR>, do not have to be closed. Some HTML programmers prefer to close them anyway. I do not. This is simply a style difference, but one you will see throughout this book.

Special Features

Each chap ter takes you step by step through sev eral exam ples show ing the tech niques that chap ter focuses on. The end of each chap ter has a sec tion called Antique Book store Pro ject. As you read through the book, this sec tion will grad u ally build a com plete web site for a fic ti tious antique book store. It is my hope that this will give you a good feel for the pro cess of devel op ing a commer cial web site. The entire code for the fin ished web site is on the com pan ion CD.

The CD

The CD that comes with this book is full of good ies. All the chap ter exam ples are on there ready to run. The com plete Antique Book store pro ject is also on the CD. Finally there is a folder full of extra sam ples you can use in your own web sites. I strongly rec om mend you browse the CD and use the resources pro vided for you.

Chapter 1

HTML Primer

This chap ter is pro vided to give you a basic work ing knowl edge of HTML. For those read ers who lack this knowl edge or feel that they may need a refresher, this chap ter is essen tial to fol low ing the rest of the book. Expe ri enced HTML pro gram mers, how ever, should feel free to skip this chap ter. JavaScript is a script ing lan guage that is embed ded into HTML doc u ments in order to add sig nif i cant func tion al ity to those web pages. For this rea son a work ing knowledge of HTML is essen tial to under stand ing and using JavaScript. I will be intro duc ing some inter est ing HTML tags as we go through the book; how ever, I feel that get ting you up to speed with a basic knowl edge is crit i cal.

HTML, or Hyper text Markup Lan guage, is a rel a tively sim ple markup language that web brows ers can use to dis play web pages. You can write HTML code in any text edi tor. I per son ally use Win dows Note pad. Just remem ber when you save the file to save it as an .htm or .html file. The browser rec ognizes files with the .htm and .html exten sions and will look in them for valid HTML code. HTML has had a long his tory and has gone through a num ber of revi sions. Each suc ces sive revi sion adds more func tion al ity to HTML, and with the cur rent ver sion (as of this writ ing) of HTML (Ver sion 4.0), it is a very pow er ful lan guage that can take some time to learn. For tu nately, most work on web pages can be done with just the essen tials of HTML, and that is what this chap ter will teach you.

NOTE: Many tags in HTML must be closed. For exam ple, the <B> tag must be closed with </B>. How ever, some tags, such as <TD> and <TR>, do not have to be closed. Some HTML pro gram mers pre fer to close them any way. I do not. This is sim ply a style dif fer ence, but one you will see through out this book.

The first ques tion is how do we get the web browser to know that our doc u-ment has HTML codes for it to read? HTML code is com posed of *tags* that let the browser know what is to be done with cer tain text. At the begin ning of your doc u ment you place the com mand <HTML> and at the end you put </HTML>; the web browser will know that the codes in between are sup-posed to be HTML.

```
<HTML>
   put HTML code here
</HTML>
```

You have to admit that this is pretty sim ple. But this web page won't do much of any thing at all. So let's do the oblig a tory "Hello World" sam ple that every pro gram ming book starts off with. It will show you how to do text and some basic HTML.

Example 1-1

```
<HTML>
<HEAD>
   <TITLE>My First HTML Page</TITLE>
</HEAD>
<BODY>
<P><CENTER>
<B><FONT SIZE="+2">Hello World</FONT></B>
</CENTER>
</BODY>
</HTML>
```

Believe it or not this lit tle snip pet shows you most of what you need to know about HTML. To begin with, note that every thing is con tained between the <HTML> and </HTML> tags. These two com mands define the begin ning and the end of the HTML doc u ment. The web browser will ignore any items out side these com mands. Next we have a sec tion that is con tained between the <HEAD> and </HEAD> com mands. This is the header por tion of your HTML doc u ment. The <TITLE> and </TITLE> com mands con tain the title that will actu ally appear in the title bar of your browser. A lot more can be done in the HEAD sec tion, but that will be addressed as we work our way through this book.

Then we have the <BODY> </BODY> com mands. As you might have guessed this is the body of your HTML doc u ment. This is where most of your web page's code is going to go. Now inside the body sec tion we have some text and some addi tional text that defines how the text will appear in the

browser. The <P> com mand defines the begin ning and the end of a paragraph. The <B> and </B> com mands tell the browser to make what ever text is between them bold. <FONT SIZE="+2"> tells the browser how big the text should be (there are a vari ety of meth ods for doing this, as we shall see). The </FONT> com mand ends the font sec tion of the HTML code. If you entered the HTML code cor rectly, then you should be able to view your web page in any browser and see an image much like that in Fig ure 1-1.

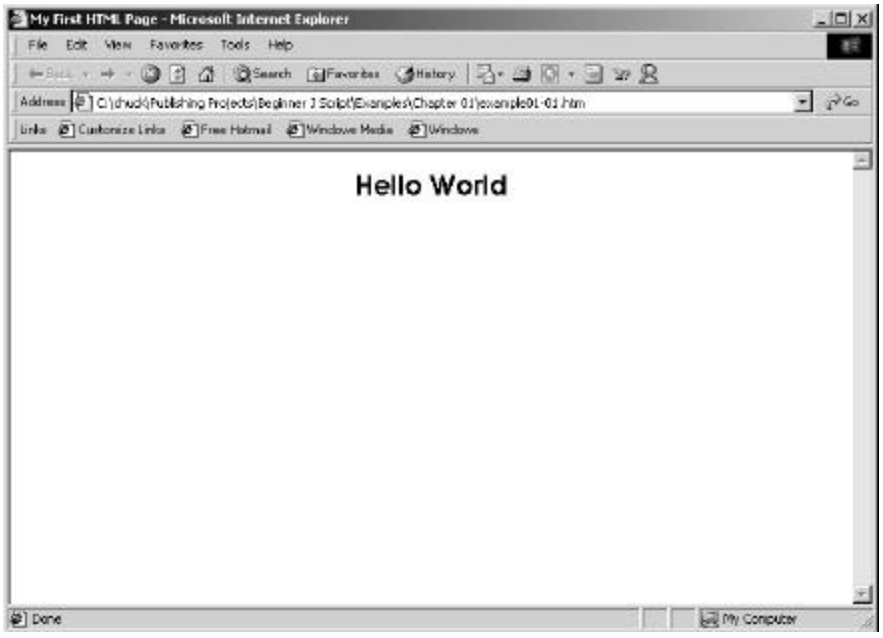


Fig ure 1-1

By now I trust you have noticed a pat tern. All the com mands have an open ing com mand and a clos ing com mand. Well, this is true for all but a very few HTML com mands. Just remem ber this rule: You close the com mands in opposite order of how you opened them. Notice in Exam ple 1-1 I opened the com mands before the text like this: <P><CENTER><B><FONT SIZE="+2">, and then closed them like this: </FONT></B></CENTER></P>. This is impor tant to remem ber. You can think of this as "back ing out" of your com mands.

Images and Hyperlinks

What we have so far gives you a very sim ple web page that dis plays one phrase in bold text. Admit tedly, this is not very impres sive, but if you under-stand the con cepts involved with using these HTML com mands then you con cep tu ally under stand HTML. Now let's expand your knowl edge of HTML. Usually web pages con tain more than sim ply a title and some text. Other items you might put in a web page would include images and links to other web pages. Placing an image on an HTML doc u ment is rather sim ple:

```
<IMG SRC="imagepath\imagename" WIDTH=52 HEIGHT=88 ALIGN=bottom>
```

You sim ply pro vide the path to the image and the name of the image, includ ing its file exten sion (such as .gif, .bmp, .jpg, etc.). The other prop er ties in this com mand allow you to alter the place ment and size of the image. You can alter its width and height as well as its align ment.

You will also note that when you first place an image on an HTML page it has a bor der around it. You can get rid of this by add ing BORDER = 0 into the tag, as in this exam ple:

```
<IMG SRC="somepic.gif" BORDER =0>
```

Putting a hyperlink to another web site or to an e-mail address is just as simple:

```
<A HREF="http://www.wordware.com">
```

This link will con nect to the URL (uni form resource loca tor) con tained inside the quo ta tion marks. In order to use this meth od ol ogy to cre ate an e-mail link sim ply use this:

```
<A HREF="mailto:sombody@somemail.com">
```

You sim ply have to change the "http://" por tion to "mailto:". Notice that all three of the pre ced ing meth ods have one thing in com mon. They do not close the com mand in the typ i cal man ner that other HTML com mands are closed. Now let's exam ine the source code for a sim ple but com plete HTML document:

Example 1-2

```
<HTML>
<HEAD>
   <TITLE>Test HTML Page</TITLE>
</HEAD>
<BODY BGCOLOR="blue">
```

```
<P>
<CENTER><B><FONT SIZE="+2">My First Web Page </FONT></B></CENTER>
<P>I am learning HTML !. I <B><I>LOVE</I></B> HTML!
<P><CENTER><IMG SRC="advscript.jpg" ></CENTER>
<P>
<P><CENTER>You can email me  at</CENTER>
<P>
<CENTER><A HREF="mailto:myemail@someemail.com">Email ME</A>
</CENTER><
<P><CENTER>Or go to this publisher's Web Site </CENTER>
<P><CENTER><A HREF="http://www.wordware.com">WordWare Publishing</A></CENTER>
</BODY>
</HTML>
```

First a few clar i fi ca tions. You should note at the begin ning a new com mand:

```
<BODY BGCOLOR="blue">
```

You can change the back ground color of your page using this com mand and any stan dard color. You can also set a back ground image for your HTML doc u ment with a sim i lar com mand:

```
<BODY background="mypicture.gif">
```

The advscript.jpg image is stored on the com pan ion CD in a folder named Chap ter 1. If you entered the code prop erly and used the image located on the CD, your web page should look some thing like Fig ure 1-2.

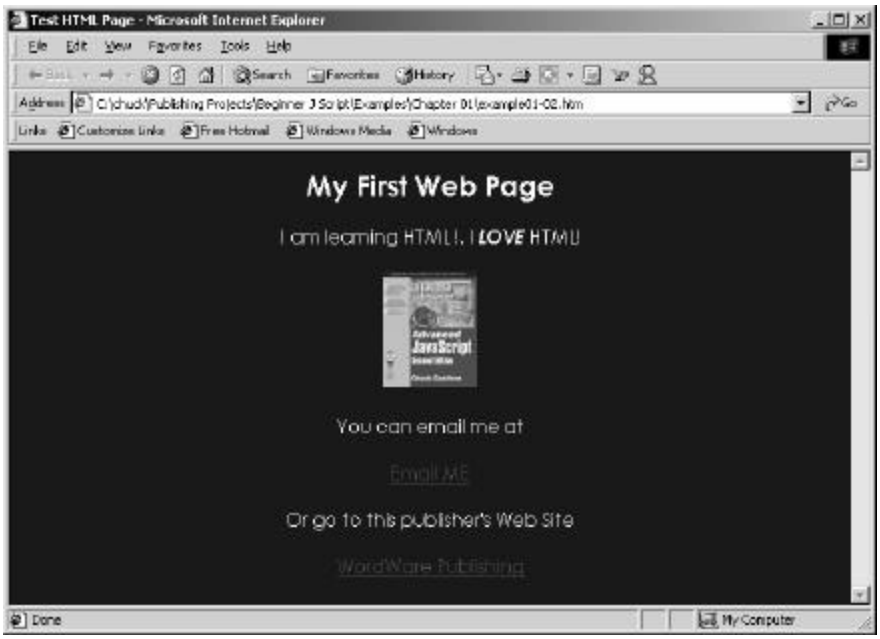


Fig ure 1-2

Now I will be the first to admit that this sam ple web page is very triv ial. But it does con tain the basics of HTML. With the mate rial we have cov ered so far you can dis play images, texts, links, e-mail links, back ground col ors, and background images. Not too bad for just a few short pages.

Colors and Backgrounds

Let's exam ine a few other sim ple items we can add to our HTML doc u ments. The first is alter ing text color. You can set the default text color for the entire doc u ment and you can alter the color of spe cific text. You alter the default text color for the entire doc u ment using a tech nique very sim i lar to the one used to alter the back ground color:

```
<BODY TEXT="blue">
```

This text sim ply tells the browser that unless oth er wise spec i fied, all text in this doc u ment should be blue. In addi tion to chang ing the default color of all text in a doc u ment you may wish to sim ply change the color of a spe cific section of text. This is fairly easy to do as well. Instead of using the BODY TEXT com mand we use the FONT com mand:

```
<FONT COLOR="red">This is red text</FONT>
```

This, like the other color com mands, can be used with any stan dard color. There are a wide vari ety of tags you can use to alter the appear ance and behav ior of text and images. Just a few oth ers for you to con sider would be the <BLINK> </BLINK> tag which, as the name implies, causes the text to blink (this is only sup ported by Netscape and will not work in Internet Explorer). Another exam ple is <STRIKE> </STRIKE>, which causes the text to appear with a line through it, a strike through. The tags we have covered so far are enough to allow you to accom plish what you need in HTML.

Tables

The next HTML com mand we are going to exam ine is the table. They are a very good way to orga nize data on your web page. You can use tables with or with out a bor der. I will explain the var i ous rea sons to use one method or the other.

First I will show you how to cre ate a table with a bor der:

Example 1-3

```
<TABLE BORDER=1>
    <TR>
      <TD>
         <P>This
         <P>Is a
   <TR>
      <TD>
         <P>Table
       <TD>
              <P>With a border
</TABLE>
```

By now you should be able to rec og nize that the <TABLE> and </TABLE> tags actu ally con tain the table. Each <TR> tag des ig nates another row in the table. The <TD> </TD> tags cre ate a cell within that row (TD refers to table data). Using those three tags you can cre ate a table with any num ber of rows or col umns you wish. Notice that the first line of this code has the BORDER prop erty set to 1. This means the bor der has a width and is therefore vis i ble.

In some instances you may not want the bor der to show. Tables can be used sim ply to hold images and text in rel a tive posi tions. In cases such as this you may not wish to have a vis i ble bor der. Below is an exam ple of a table whose bor ders will not show.

Example 1-4

```
<P><TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
   <TR>
      <TD>
         <P>This
     <TD>
         <P>is a
   <TR>
      <TD>
```

```
            <P>Table
        <TD>
            <P>With no borders or padding
</TABLE>
```

Notice that the BORDER, CELLPADDING, and CELLSPACING prop er ties are all set to 0. This causes the table itself to not be dis played. How ever, the con tents of the table will dis play. You should also notice that in both exam ples I have placed text in each cell.

Fig ure 1-3 shows two tables, one with a bor der and one with out.

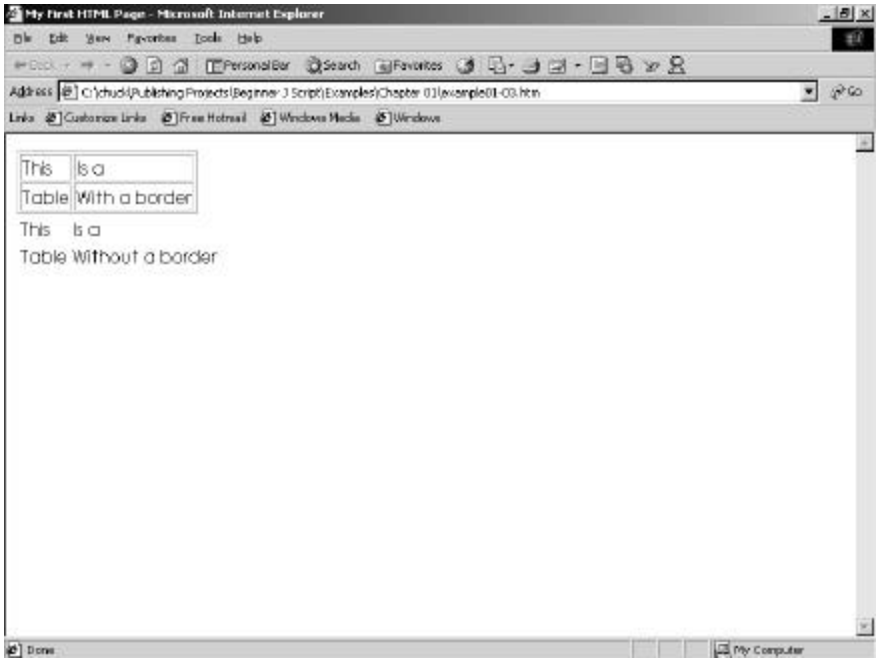


Figure 1-3

Since the entire pur pose of this book is to teach you JavaScript, the obvi ous ques tion on your mind should be “How do I insert scripts into HTML?” Well, for tu nately, that is not par tic u larly dif fi cult. The browser deals with script code (includ ing JavaScript) like it han dles HTML code, scan ning the source from left to right and from top to bot tom. JavaScript has “tags” of its own by which it instructs the browser and deter mines the lay out. How ever, JavaScript is not read exactly like HTML. In HTML, the browser acts imme di ately accord ing to the ele ments it rec og nizes. Not all JavaScript code refers to actions that take place while the page is load ing. Some parts are just kept in mem ory until they are called. For instance, if you write a func tion and do not call it, the browser does not do any thing with it. This part of the script stays in

mem ory and can be invoked later. But the real ques tion you are likely won dering about before you pro ceed with the rest of this book and learn ing JavaScript is, how do you put scripts into your HTML doc u ments. Any script can be placed in a doc u ment very eas ily by encas ing the script inside of two commands:

```
<SCRIPT LANGUAGE = "whateverscriptlanguageyouareusing">
```

and

```
</SCRIPT>
```

There is a vari ety of script ing lan guages avail able for the web includ ing VBScript (based on the Visual Basic pro gram ming lan guage), CGI, and of course, JavaScript. Below are two exam ples of JavaScripts inserted into HTML code.

```
<SCRIPT LANGUAGE="JavaScript">
document.write("<IMG SRC='img4-2.gif' ALT='pencil icon' HEIGHT=71 WIDTH=53>")
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
function makeDialogBox()
{
    alert("Wonderful!")
}
</SCRIPT>
```

Lists

It is com mon to pres ent data in lists. With HTML you have access to a vari ety of types of lists. The first we will dis cuss is the unor dered list shown below.

```
<UL>
     <LI> First Item
     <LI> Second Item
 </UL>
```

The <UL> and </UL> tags define the code the lies between them as being part of an unor dered list. The <LI> tags iden tify list items. An unor dered list item will sim ply appear as a bulleted item, as shown in the fol low ing fig ure.

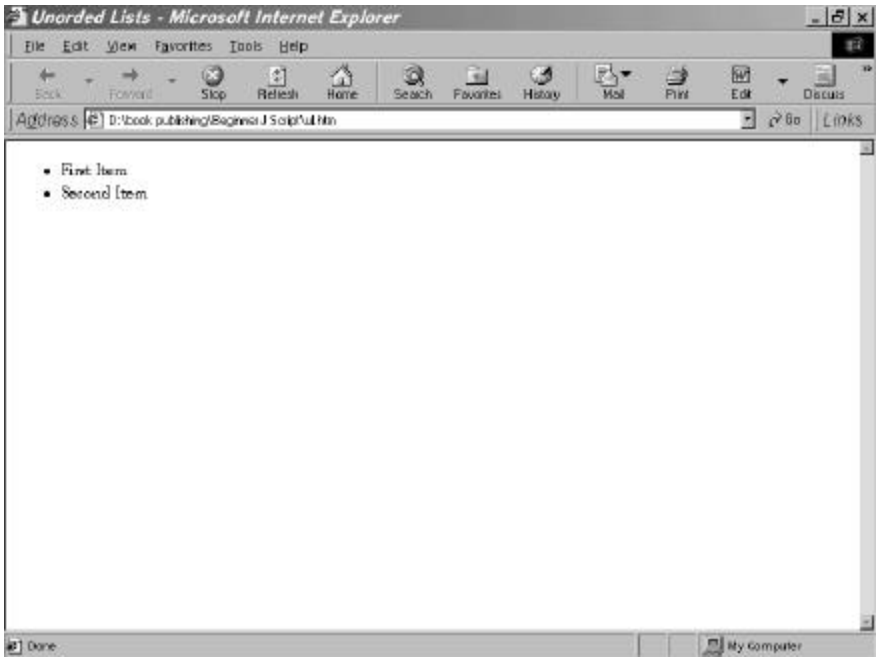


Figure 1-4

An ordered list is not much different. The <LI> list item stays the same. But the <UL> is going to change some what.

```
<OL type = I>
     <LI> First Item
     <LI> Second Item
</OL>
```

The "type =" portion of the tag tells the browser what type of list this is. We used a capital I in our example, which will give you capital roman numerals for your list items.

Figure 1-5

The fol low ing table shows all the types of ordered lists and how they appear in your browser.

Type = I	I. First Item II. Second Item
Type = i	i. First Item ii. Second Item
Type = 1	1. First Item 2. Second Item
Type = a	a. First Item b. Second Item
Type = A	A. First Item B. Second Item

Marquee

A very fas ci nat ing item you can add to your web page is the mar quee. A scrolling mar quee takes a mes sage or an image and scrolls it across the screen. The basic for mat is this:

```
<MARQUEE LOOP = INFINITE> Hey this is really cool </MARQUEE>
```

In addi tion to text, you can also place an image in the mar quee to scroll across the screen:

```
<MARQUEE LOOP = INFINITE> <IMG SRC = "mypic.gif"></MARQUEE>
```

You can also change the direc tion the mar quee moves in. The <DIRECTION> tag will tell the mar quee which direc tion to scroll to, not from.

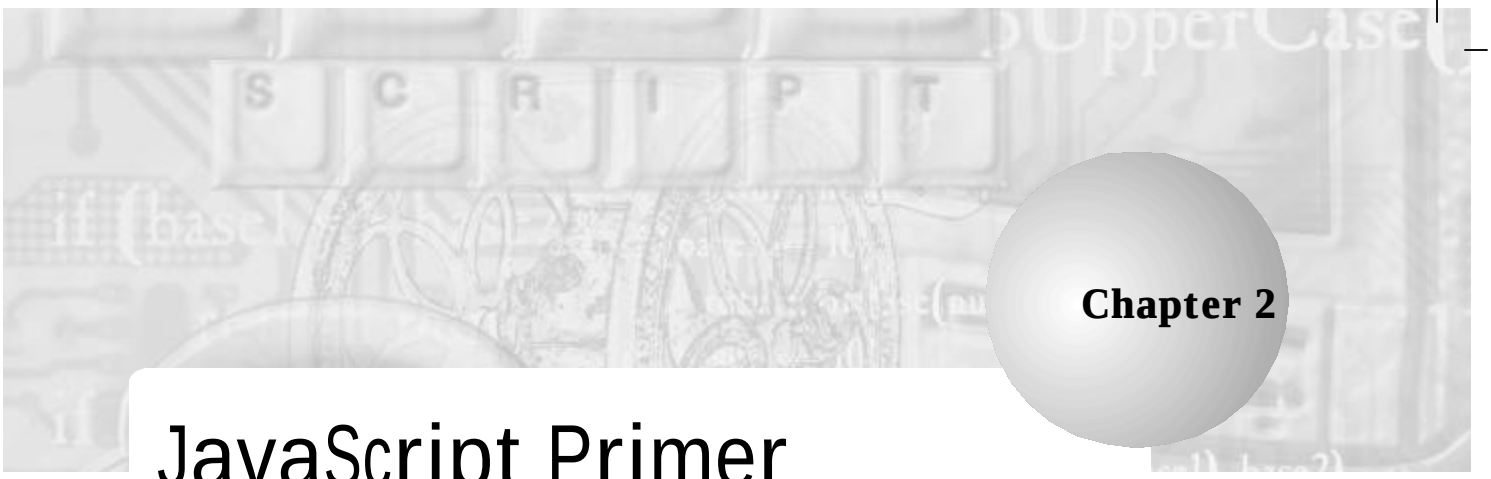
```
<MARQEE LOOP = INFINITE DIRECTION = RIGHT>Hey this is a cool marquee </MARQUEE>
```

You can choose from the fol low ing direc tions: Left, Right, Up, and Down. Marquees pro vide an inter est ing and rel a tively easy way to dis play eye-catching infor ma tion on your web page.

Summary

Let me stress that what I have just cov ered with you are sim ply the essen tials of writ ing HTML code. You can find a num ber of books that will go into HTML in more depth than this.

There are also a vari ety of soft ware pack ages that will do the HTML code for you. You sim ply cre ate a doc u ment just as you would in any word pro ces sor and the soft ware will cre ate the under ly ing HTML code. Two very inex pensive options that I rec om mend for basic HTML cod ing are Claris Works Home Page (about $25) and Netscape Com poser, which comes with Netscape Commu ni ca tor and is abso lutely free of charge.



Chapter 2

JavaScript Primer

This book will teach you JavaScript by walk ing you step by step through the pro duc tion of a com plete web site. How ever, to get you a jump start it is essen tial that you first have a basic grasp of JavaScript. The goal of this chapter is to intro duce you to the basic con cepts of JavaScript. Each of the suc ces sive chap ters dis cusses a par tic u lar facet of JavaScript with sev eral exam ples. Just before the sum mary of each of the fol low ing chap ters is a section where we use the mate rial cov ered in that chap ter for the web site we are building.

This chap ter will take you into the tech ni cal details of JavaScript. It is pos si ble to read the later chap ters and use them with out read ing this chap ter. How ever, at some point you will want to review this chap ter. Its con tents explain the intri ca cies of how JavaScript works. Even if you skip this chap ter, I rec ommend you come back to it when you have read the first 10 chap ters of the book.

Essential Tools

HTML files are plain text files. There fore, in order to add JavaScript to an exist ing HTML doc u ment, you need a text edi tor. Note pad is an excel lent choice for a sim ple text edi tor, if for no other rea son than it's free with Windows. Most oper at ing sys tems includ ing Win dows, Macintosh, and Linux come with a free text edi tor.

You can use var i ous edi tors for your JavaScript as well as your HTML. However, many expe ri enced web devel op ers sim ply use a stan dard text edi tor that usu ally comes free with their oper at ing sys tem. If you are using Win dows, you already have Note pad and WordPad on your PC; those are both fine for doing HTML and JavaScript. The advan tage of using stan dard text edi tors is that they are free. The dis ad van tage is that they do not offer any debug ging tools to

help you with the JavaScript. I per son ally pre fer sim ply using Note pad for my HTML and JavaScript devel op ment.

Choosing a Browser

Besides the basic pro gram ming tool, you need to be able to run your scripts and view their out put. It does n't really mat ter which browser you choose. The lat est ver sions of Netscape and Microsoft Internet Explorer both fully sup port JavaScript. Com pat i bil ity should only be a seri ous prob lem with older ver sions of brows ers. Since both brows ers are avail able as a free down load, it is prob a-bly advis able that you down load a copy of both so that you can test your scripts in both.

Basic JavaScript Structure

In order to run cli ent-side JavaScript, you must embed the code in the HTML doc u ment. How ever, you can not place JavaScript state ments in the source code in just any loca tion. There are basi cally two meth ods to embed JavaScript scripts in HTML:

- As state ments and func tions using the <SCRIPT> </SCRIPT> tags.
- As short state ments resem bling URLs.

The <SCRIPT> Tag

This tag defines in HTML that a script fol lows. You can declare a script that is directly writ ten in the HTML code, or in a sep a rate file.

Internal Scripts

The <SCRIPT> tag is used to enclose JavaScript code in HTML doc u ments. This is the most com mon way to include sim ple JavaScripts and it is the method we will use for most of this book. Here is the gen eral syn tax:

```
<SCRIPT LANGUAGE="JavaScript">
[JavaScript Statements...]
</SCRIPT>
```

The <SCRIPT LANGUAGE="JavaScript"> tag acts like all other HTML tags. Notice that it must be fol lowed by its clos ing coun ter part, </SCRIPT>. Every state ment you put between the two tags will be inter preted as JavaScript code. This is prob a bly the most com mon method for insert ing JavaScript into HTML documents.

The LANGUAGE attrib ute is used to spec ify the script ing lan guage. At present, the <SCRIPT> tag sup ports var i ous lan guages includ ing JavaScript and VBScript. Java-Script is the default script ing lan guage, so the LANGUAGE def i ni tion is not required. How ever, I rec om mend that you still use it for the sake of clar ity. When the browser comes across the pre cise name of the language, it loads the built-in JavaScript inter preter and then inter prets the script.

JavaScript is case sen si tive, but HTML is not. It does not mat ter whether you write <SCRIPT> or <script>, but try to be con sis tent. Be sure to write the name of the lan guage prop erly. Per sonally I use all cap i tals in my HTML; even though it does not mat ter to the browser, it does to the per son read ing my HTML code. You will find that many web devel op ers fol low this con ven tion. Remem ber that HTML and JavaScript are no dif fer ent from any pro gram ming lan guage in that you should always strive to make sure your code is read able.

External Scripts

There is a SRC attrib ute for the <SCRIPT> tag, which enables the use of exter nal scripts. This means you can place your script in a sep a rate file and include it using the SRC attrib ute. Exter nal scripts are use ful when you have lengthy or com plex scripts and you feel that includ ing them directly in your HTML code would make for dif fi cult read ing. I will be using exter nal scripts later in this book.

Placing JavaScript Code

When you want to place the script some where in the HTML doc u ment, you need to choose an appro pri ate place to put it. Tech nically, you may place it anywhere between the <HTML> and </HTML> tags that enclose the whole doc u ment. Actually, the two pos si bil i ties are the <HEAD>…</HEAD> portion and the <BODY>…</BODY> por tion. Because the <HEAD>…</HEAD> por tion is eval u ated first, many devel op ers choose to place their JavaScript here. A sin gle HTML doc u ment may con tain any num ber of scripts. You can place some of the scripts in the <HEAD>…</HEAD> por tion, and oth ers in the <BODY>…</BODY> por tion of the page. The choice of where to place them should fol low some logic based on when you wish the script to exe cute. Script placed in the <HEAD> sec tion will be exe cuted as soon as the web page is loaded. The fol low ing code dem on strates this:

```
<HTML>
<HEAD>
<TITLE>Using Multiple scripts</TITLE>
<SCRIPT LANGUAGE="JavaScript">
[JavaScript statements...]
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
[JavaScript statements...]
</SCRIPT>
</HEAD>
<BODY>
<H1>This is another script</H1>
<SCRIPT LANGUAGE="JavaScript">
[JavaScript statements...]
</SCRIPT>
</BODY>
</HTML>
```

JavaScript Conventions

There are sev eral con ven tions used to make JavaScript code more under stand-able. Some of these are dis cussed in this sec tion.

Using the Semicolon

The JavaScript inter preter does not pay any atten tion to car riage return char-ac ters in the source. It is pos si ble to put numer ous state ments on the same line, but you must sep a rate them with a semi co lon (;). You can also add a semi-co lon at the end of a state ment that occu pies its own line, but it is not nec es sary. Take a look at the fol low ing state ments:

```
document.write("Hello"); alert("Good bye")
```

```
document.write("Hello")
alert("Good bye")
```

```
document.write("Hello");
alert("Good bye");
```

All three sets are legal, and their results are iden ti cal. Let me stress that the semi co lons are not needed; I just men tion them because pro gram mers whose back ground is C, C++, or Java will be used to end ing their state ments with a semi co lon. Although the semi co lon is allowed, I per son ally never use it, since its use is super flu ous in JavaScript.

Case Sensitivity

It was men tioned pre vi ously in this chap ter that JavaScript is a case-sensitive lan guage. This fact applies to all aspects of the lan guage, includ ing vari able names (*identifiers*), func tions, and meth ods (dis cussed later). The state ment doc u ment.write(), for exam ple, is legal, but doc u ment.Write() is not.

Comments

Com ments are an impor tant con cept in all lan guages, includ ing in JavaScript. They help make pro grams sim ple and easy to under stand. Com ments are messages that can be put into pro grams at var i ous points with out affect ing the results. There are two dif fer ent types of com ments in JavaScript:

- Sin gle-line com ments are com ments that do not exceed the length of one line. These com ments begin with a dou ble slash (//).
- Mul ti ple-line com ments are com ments that exceed the length of one line. Therefore, they require a sign that spec i fies the begin ning of the com ment, and another sign that spec i fies the end of the com ment. The open ing part is /* and the clos ing part is */.

```
// single-line comment
/* line 1 of the multi line comment
   line 2 of the multi line comment
   line 3 of the multi line comment */
```

Both types of com ments are iden ti cal to the com ments in Java, C, and C++.

Com ments are an essen tial part of any pro gram ming lan guage. If you work in a team envi ron ment, com ments will allow oth ers to clearly under stand the inten tions of your code. I would also add that if you review your own code several months after writ ing it, you are unlikely to remem ber exactly what you where plan ning. Although some pro gram mers will state that very well-written code needs few or no com ments, I strongly dis agree with this. I per son ally advo cate a very lib eral use of com ments in your code. Even if you "over-comment," abso lutely no harm is done.

Using Quotes

In JavaScript, you often use quotes to accom plish var i ous goals, such as to delimit strings. A com mon prob lem arises when using a pair of quotes inside another pair of quotes. Since the inter preter must rec og nize each set of quotes in order to pair them cor rectly, the cre ators of JavaScript made it pos si ble to use two dif fer ent types of quotes: dou ble quotes (") and sin gle quotes ('). If you need only one set of quotes, you can choose either of them as long as you ter mi nate the text with the same type of quote you used to open it. If you use

quotes improp erly, you will get a JavaScript error: “unterminated string literal.” You must make cer tain that you alter nate quotes prop erly:

```
document.write("<IMG SRC='cool.gif'>")
```

Your First Script

Now that you have read about the basic con cepts behind JavaScript, I think it is time to actu ally write a lit tle JavaScript. You are prob a bly quite bored with all this back ground infor ma tion and eager to see some action.

First of all, launch your text edi tor. Type Exam ple 2-1 in the text edi tor and save it under the name Hello.htm. Make sure the name ends with either the .htm or .html exten sion, or your browser will not rec og nize this as an HTML doc u ment. It is imper a tive, how ever, that as you type in this exam ple you make every attempt to fully under stand what it is you are typ ing. Next, launch the browser. Since the file is local, you do not need to be con nected to the Internet to view the results. Now, load the file from the browser’s menu. That’s all there is to it. You can start enjoy ing JavaScript.

The fol low ing script is inter preted and exe cuted imme di ately when you load the page con tain ing it.

Example 2-1

```
<HTML>
<HEAD>
<TITLE>Hello World.</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=”JavaScript”>
<!—hide code from old browsers
document.write(“<H1>Hello World.</H1>”)
// end code hiding
</SCRIPT>
</BODY>
</HTML>
```

If you entered the code prop erly you should see some thing in your browser sim i lar to Fig ure 2-1.

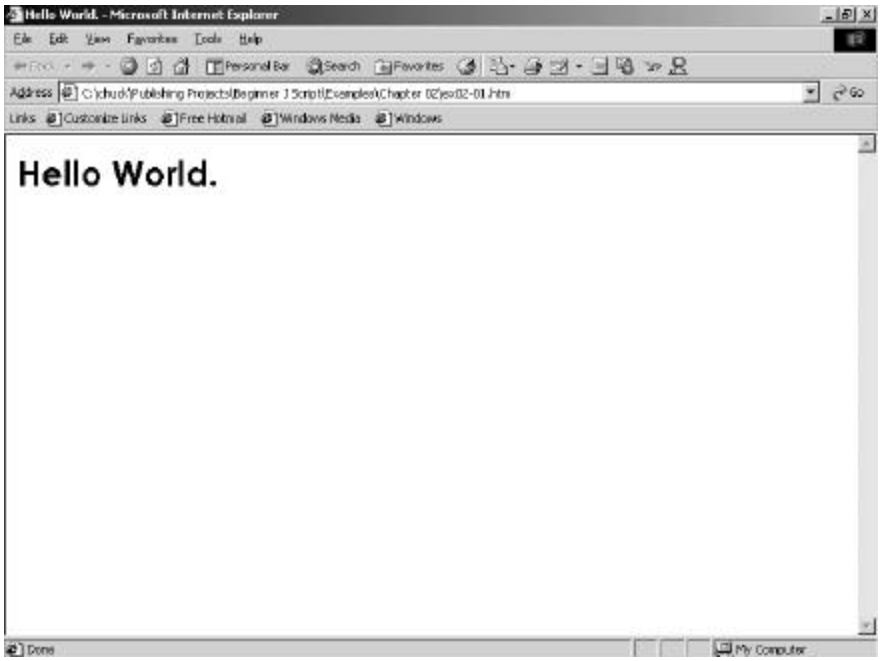


Fig ure 2-1

This exam ple is OK, but I per son ally pre fer using JavaScript alert boxes. So let us re-write Exam ple 2-1 using an alert box:

Ex am ple 2-2

```
<HTML>
<HEAD>
<TITLE>Hello Again!</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!—hide code from old browsers
  alert("Hello World")
// end code hiding
</SCRIPT>
</BODY>
</HTML>
```

These two, very basic JavaScripts should give you a feel for the lan guage, and hope fully whet your appe tite to learn more. How ever, before we can delve into the excit ing world of build ing dynamic web sites with JavaScript, we need to cover the more mun dane fun da men tals of the lan guage.

Fundamental JavaScript Concepts

JavaScript is com posed of the same basic ele ments as all pro gram ming languages. It has vari ables, which hold data, it has expres sions or state ments, and it has func tions. Each of these build ing blocks are used to con struct your JavaScript.

Data Types and Variables in JavaScript

Vari ables are essen tially named seg ments of mem ory set aside to hold data of a cer tain type. These types are referred to as data types. When you cre ate a vari able you are sim ply allo cated a space in mem ory for the data type of the vari able you declare. It is impor tant to use the proper data type to store your infor ma tion. For exam ple, a num ber is a type of infor ma tion that JavaScript rec og nizes. Compared to most pro gram ming lan guages, JavaScript has a small num ber of data types. There are only four dif fer ent data types in JavaScript: num bers, strings, Boolean, and null val ues. As opposed to other lan guages, a vari able data type is not declared explic itly but rather implic itly accord ing to its ini tial value assign ment. Also unique to JavaScript is there is no explicit dis tinc tion between inte ger and real-valued num bers.

JavaScript is a loosely typed lan guage. The only key word for declar ing a JavaScript vari able is var. The actual data type used depends on the ini tial value you assign to the vari able. Con sider the fol low ing exam ples:

```
var LastName = "Smith"
var AccountNumber = 1111
```

In the first exam ple, the data type of LastName is string, sim ply because that is the ini tial value assigned to that vari able. In the sec ond exam ple, the data type of AccountNumber is num ber.

This method of vari able dec la ra tion is in stark con trast with strongly typed lan guages such as C or Java. In strongly typed lan guages the vari able must be cre ated of a spe cific type.

```
int myint:
boolean myboolean;
```

There are a few rules to observe when cre at ing vari ables:

- All other char ac ters can be let ters, under scores, or dig its (0-9).
- An iden ti fier can not be one of the lan guage's reserved words. Reserved words are basi cally the words that make up the JavaScript lan guage.

- JavaScript is case sen si tive (upper case let ters are dis tinct from low er case let ters). For exam ple, coun ter, Coun ter, and COUNTER are the names of three dif fer ent vari ables. Avoid using such sim i lar iden ti fi ers in the same script.

The fol low ing vari able names are legal:

```
loopcounter
_employeename
_123456789_
account_number
Number16
```

but the fol low ing ones are ille gal:

```
with                    // reserved word
^mystring               // first character is illegal
411information          // cannot start with a digit
04-825-6408             // first character is illegal
                        // "-" is an illegal character
***important***         // * is not a legal character
```

The action of assign ing an ini tial value to a vari able is called *initialization*. You give the vari able a value using the *assign mentoperator*—the equal sign:

var *variableName* = *initialValue*. You only need to use the var key word when you cre ate the vari able. When you want to refer to the vari able, you only use its name. Assign a value to a vari able (after it has been declared) in the fol low-ing fash ion:

```
variableName = anyValue
```

You use var only once per vari able. A global vari able can be cre ated sim ply by assign ing it a value with out the var key word. Local vari ables inside func tions, on the other hand, must be declared with the var key word. As in many other pro gram ming lan guages, JavaScript allows you to declare numer ous vari ables in the same state ment, using a comma to sep a rate them:

```
var variableName1 = initialValue1, variableName2 = initialValue2, …
```

JavaScript Operators

Every pro gram ming lan guage has oper a tors. An oper a tor is sim ply a sym bol that tells the com piler (or inter preter) to per form a cer tain action. The basic arith me tic oper a tors are com mon to most pro gram ming lan guages. These are addi tion (+), sub trac tion (–), mul ti pli ca tion (*), and divi sion (/). These should be very famil iar to most peo ple. The order of pre ce dence of oper a tors fol lows the stan dard math e mat i cal rules of mul ti pli ca tion, divi sion, addi tion, and

sub trac tion. How ever, when your code has mul ti ple oper a tions in a sin gle line it is usu ally a good idea to use paren the ses to clar ify what you want to occur; 3 * 4/2 + 1 can be ambig u ous, whereas 3 * ((4/2) + 1) is very clear.

C, C++, and Java pro gram mers will already be famil iar with the incre ment and dec re ment oper a tors, while other pro gram mers may not be. The increment oper a tor is done by plac ing two plus signs after a vari able, such as this:

```
Var somenumber
Somenumber++
```

This line of code incre ments the value of somenumber by one. Had we written:

```
somenumber - -
```

it would have decreased the value by one.

It is very impor tant that you real ize that where you place the incre ment and dec re ment oper a tors is crit i cal. If you place the incre ment oper a tor after a vari able such as:

```
Var somenumber = 10
Var someothernumber
Someothernumber = somenumber++
```

The assign ment oper a tion will take place before the eval u a tion oper a tion. In other words, first “someothernumber” will be set equal to the value of “somenumber” then the value of “somenumber” will be incre mented. In our exam ple that means that “someothernumber” will equal 10 and somenumber will equal 11. If you wish to rewrite the state ment so that the incre ment takes place first, just repo si tion the incre ment sign:

```
Someothernumber = ++somenumber
```

In this case, somenumber is incre mented to 11 and then that value is assigned to someothernumber.

You’ve already learned how to assign a value to a vari able or to ini tial ize it using the equal assign ment oper a tor. As the fol low ing piece of code dem onstrates, you can also per form cal cu la tions when you assign a value:

```
/* 1 */ var answer
/* 2 */ answer = 4 * 2 + 9
/* 3 */ document.write(answer)
```

Line 1 includes the dec la ra tion of the vari able answer. The sec ond line shows how the vari able answer is assigned the result of a sim ple math e mat i cal expres sion. At this point, the vari able holds a value of 17. Referring to the vari able answer is the same as refer ring to the num ber 17. For this rea son, the state ment on line 3 prints the num ber 17.

Caution: A very common mistake is to use the equal sign for equality check. In Visual Basic, for example, = is an equality test operator, because the basic assignment operator of the language is :=. However, in JavaScript, like in C++ and Java, = (the equal sign operator) is an assignment operator, while == (two equal signs) is an equality test operator.

JavaScript Statements

Now that we have thor oughly exam ined data types, let's look at state ments. A *statement* is sim ply a line of code that per forms some spe cific task or action. For exam ple, all of the fol low ing are state ments:

```
myAge = 32
for(x=1;x<10,x++)
myname= "Chuck"
```

Multiple Statements

The JavaScript inter preter accepts mul ti ple state ments on the same line. If you choose to use this method, you must sep a rate the state ments with semi-co lons (;). Note that this is the only place in JavaScript where you are required to use the semi co lon, and, in my opin ion, the only place you should use it. The last state ment of the line does not have to be fol lowed by a semi co lon. Such a line looks like this:

```
statement1; statement2; statement3; ...
```

The browser inter prets these state ments as if they were listed on sep a rate lines:

```
statement1
statement2
statement3
```

Although this is pos si ble with JavaScript I cer tainly do not rec om mend it. Placing mul ti ple state ments on a sin gle line makes for very unread able code.

Nested Statements

A com mand block is a unit of state ments enclosed by curly braces. It is very impor tant to under stand that a block should be used as a sin gle state ment. The state ments inside the block are called *nested state ments*:

```
{
     nested statement1
     nested statement2
     nested statement3
}
```

A loop that includes many state ments is actu ally one state ment with many nested state ments. This rule applies to func tions, if-else state ments, and other lan guage ele ments.

JavaScript Expressions

Now that you know how to cre ate a vari able, you need to know how to use it. As men tioned ear lier, vari ables hold val ues of dif fer ent types. What does "hold ing a value" mean? This term refers to expres sion *evaluation*. A vari able always eval u ates to its value. When you per form an oper a tion on a vari able, you are actu ally per form ing the oper a tion on the cur rent value asso ci ated with the vari able. Let's assume you cre ated a vari able named firstNumber using the following statement:

```
var firstNumber = 120 // declaration and initialization
```

At this point, if you refer to the vari able firstNumber, its value, 120, is returned. That is, firstNumber is eval u ated to 120. The fol low ing state ment out lines an eval u a tion of firstNumber:

```
secondNumber = firstNumber * 6
```

The secondNumber vari able now holds the value 720, because firstNumber eval u ates to 120. Bear in mind that no link between the mem ory loca tions of the vari ables is estab lished. There fore, secondNumber now holds a value of 720, which does not change even if the value of firstNumber changes. A vari-able can eval u ate to a value of any type.

JavaScript Function Declarations

A func tion is sim ply a group of related state ments that per form some com mon goal and are grouped together under some com mon func tion name. Just like vari ables, you must define a func tion before you can call it.

Func tions are defined using the key word func tion, fol lowed by the name of the func tion. The same rules that apply to nam ing vari ables as apply to nam ing func tions. Since a func tion usu ally does some thing besides stor ing a value, it is com mon to include a verb in its name. The most impor tant thing to remem ber about nam ing func tions is that the name should pro vide some indi ca tion of what the func tion does. The func tion's param e ters are writ ten in brack ets after the name. Param e ters are sim ply val ues that are passed into a func tion for the func tion to pro cess. The syn tax of a func tion def i ni tion is:


```
function functionName([parameters])
{
     [statements]
}
```

Parameters are local vari ables that are assigned val ues when the func tion is called. At this point, you should always give a name to every param e ter.

In a for mal syn tax spec i fi ca tion, the square brack ets "[" and "]" usu ally denote optional ele ments. Since a func tion does not have to have param e ters or state-ments, they are both enclosed in such brack ets. The curly braces enclos ing the func tion body can be placed any where, fol low ing the param e ter sec tion. The fol low ing func tions are valid:

```
function functionName([parameters]) {[statement1]; [statement2]; …}
function functionName([parameters])
{
     [statement1]
     [statement2]
}
```

The fol low ing exam ple dem on strates a func tion dec la ra tion:

Example 2-3

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!— hide script contents from old browsers
function square(number)
{
     document.write("The call passed ",number,// the function's parameter
          " to the function.<BR>",
          number,                              // the function's parameter
          " square is ",
          number * number,
          ".<BR>")
}
```

```
// *** add function call
// end hiding contents from old browsers  —>
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Exam ple 2-3 does not print any thing to the browser's win dow, nor does it gen-er ate any other form of out put. The rea son is that the func tion is only defined in the script but never called. When the browser locates a func tion, it loads its state ments into the mem ory, ready to be exe cuted later.

Calling Functions

In order to exe cute the set of state ments located in the func tion block, you must call the func tion. The syn tax of a func tion call is:

```
functionName([arguments])
```

By add ing the state ment square(5) to Exam ple 2-2, at the spec i fied place, we call the func tion. The state ments in the func tion are exe cuted, and the fol low-ing mes sage is out put:

```
The call passed 5 to the function.
5 square is 25.
```

You can also call a func tion from within another func tion, as the fol low ing example demonstrates:

Example 2-4

```
<HTML>
<HEAD>
<TITLE>Calling a function from within another function</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!— hide script contents from old browsers
function makeBar()
{
     var output = "<HR ALIGN='left' WIDTH=400>"
     document.write(output)
}
function makeHeader(text, color, size)
{
     var output = "<FONT COLOR='" + color + "' SIZE=" +
          size + ">" + text + "</FONT>"
     document.write(output)
      makeBar()
```

```
}
makeHeader("JavaScript Examples", "red", "+4")
// end hiding contents from old browsers  -->
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```


Exam ple 2-4 sum ma rizes many of the terms dis cussed in this chap ter. It includes two func tion def i ni tions. In both func tions, the out put is assigned to a vari able (out put) and then printed to the cli ent win dow using the doc u-ment.write() method. Assigning strings to vari ables before print ing them is extremely use ful when the string is long (you want to print a lot of data). You can see the result of Exam ple 2-3 in the fol low ing image.

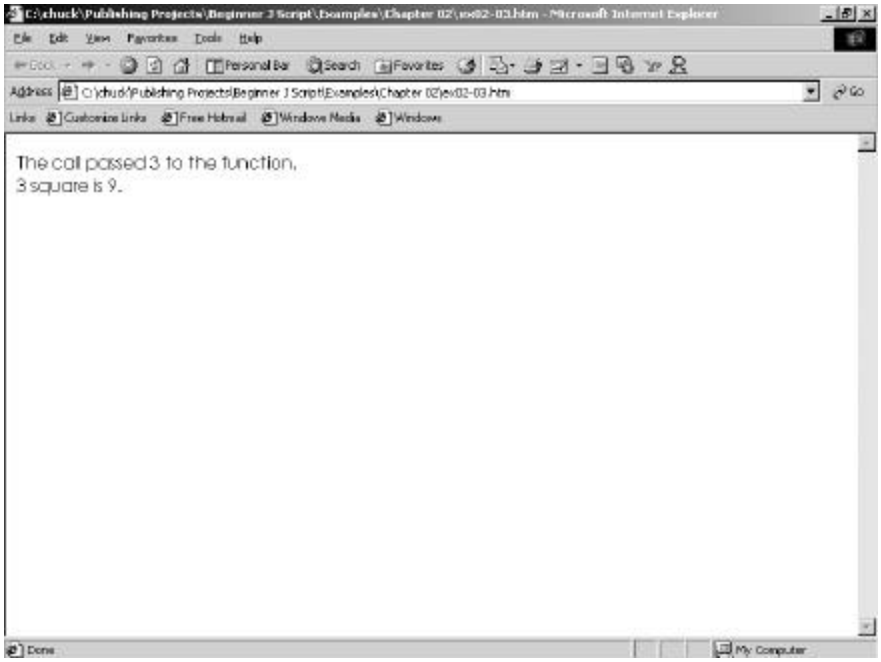


Fig ure 2-2

Dialog Boxes

JavaScript pro vides the abil ity to cre ate small win dows called *dia log boxes.* You can cre ate small alert boxes, con firm boxes, and prompt boxes. These boxes let you gen er ate out put and receive input from the user.

Alert Boxes

An alert box is the most sim ple dia log box. It enables you to dis play a short mes sage to the user in a sep a rate win dow. Take a look at the fol low ing script and its cor re spond ing out put:

```
alert("Click OK to continue...")
```

The generic form of this func tion is alert(*message*). The func tion alert() is actually a method of the win dow object. It is not nec es sary to spec ify that because win dow is the default object. The same applies to all dia log boxes.

Note: Netscape Communications Corp. implemented the "JavaScript Alert:." header for security reasons. It is used to distinguish JavaScript dialog boxes from those created by the operating system, so that the user knows what the source of the message is. JavaScript programmers cannot trick the user into doing something he might not want to do. It also disables the ability to scare the user into giving out personal information.

You can also dis play mes sages using vari ables. For exam ple:

```
var message = "Click OK to continue"
alert(message)
```

The alert box is often used to pause the exe cu tion of a script until the user approves its con tin u a tion.

Confirm Boxes

Con firm boxes are dif fer ent from alert boxes in that they eval u ate to a value based on a deci sion made by the user. Rather than a sim ple OK but ton, the con firm box includes both OK and Can cel but tons.

Like the alert box, con firm is also a method of the win dow object. This method returns a Boolean value, because there are two options. You can use con fir ma tion boxes to ask the user a yes-or-no ques tion, or to con firm an action. Here is an exam ple and its out put:

```
var reply = confirm("OK to continue?")
```

reply is assigned a true value if the user chooses OK, and false if the user selects Can cel. The generic form of this func tion is con firm(*message*).

Prompt Boxes

The prompt() method dis plays a prompt dia log box with a mes sage and an input field. You can use these boxes to receive input from the user. It is sim i lar to the con firm box, except that it returns the value of the input field, rather than true or false. Here is an exam ple:

```
var name = prompt("Enter your name:", "anonymous")
```

The method returns a value of null if the user chooses Can cel. The prompt box looks like the image shown in Fig ure 2-3.

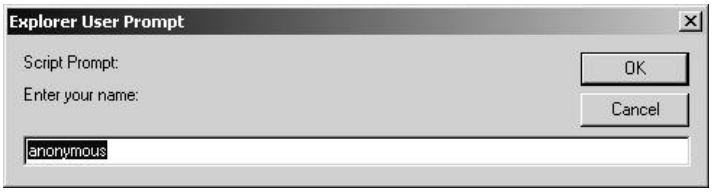


Fig ure 2-3

The value of the field is always a string. If the user enters 16 in the form, the string "16" is returned rather than the num ber 16. When you want to prompt the user for a num ber, you must con vert the input into a numeric value. JavaScript fea tures a built-in func tion that does this—parseInt(). You can use the fol low ing state ment to ask the user for a num ber:

```
var number = parseInt(prompt("Enter a number:", 0))
```

or

```
var number = prompt("Enter a number:", 0)
number = parseInt(number)
```

The generic form of this func tion is prompt(mes sage[, inputDefault]).

You can see that this func tion works by using the typeof oper a tor for test ing:

```
var number = prompt("Enter a number:", 0)
alert(number, " is a ", typeof number) // "... is a string"
number = parseInt(number)
alert(number, " is a ", typeof number) // "... is a number"
```

The input must be of a numeric type, of course (e.g., 99).

if Statement

```
if (condition)
     statement
```

The if state ment lets you put deci sion mak ing in your scripts. A script with out any deci sions does the same pro ce dure each time it is exe cuted. Such lin ear struc tures limit your scripts to sim ple algo rithms. JavaScript enables deci sion mak ing using an if state ment. if state ments asso ci ate a sin gle state ment with a true con di tion. That state ment is only exe cuted if the con di tional expres sion is true; oth er wise it is not exe cuted at all. The con di tion must eval u ate to a Boolean value: true or false. Numeric val ues are also accept able as an alter native to a Boolean con di tion. 0 is equiv a lent to false, and all other val ues are equiv a lent to true.

The if state ment asso ci ates a sin gle state ment with a true con di tion. A statement can be any thing from a sim ple doc u ment.write() to a block of state ments using curly braces ({}). Some if state ments require mul ti ple state ments, so they use a block in the fol low ing form:

```
if (condition)
{
     statement1
     statement2
     statement3
}
<HTML>
<HEAD>
<TITLE>A simple if statement</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!—
var age = parseInt(prompt("Please enter your age:", 15))
if (age > 30)
     alert("Wow, you are old enough to remember Disco!")
// —>
</SCRIPT>
</HEAD>
<BODY>
   JavaScript Example
</BODY>
</HTML>
```

This is a sim ple HTML doc u ment that includes a JavaScript script. And admittedly it is noth ing to get overly excited about. How ever, it does illus trate exactly how to place JavaScript into an HTML doc u ment. Notice that its structure is the same as that of any other HTML doc u ment. The only new con cept is the <SCRIPT> tag. I put the script in the <BODY>…</BODY> por tion of the page, though you may put it any where between the <HTML> and </HTML> tags. For now, think of doc u ment.write() as a way to print expressions to the page. write() is actu ally a method of the doc u ment object. Objects, meth ods, and prop er ties are intro duced in Chap ter 6, "The Doc u ment Object Model." The write() method sup ports any HTML syn tax. Be aware, also, that all strings must be included in quotes.

Summary

This chap ter showed you the basics of JavaScript. This, com bined with the HTML primer, should give you the foun da tional skills nec es sary to fol low the rest of this book. Make sure you are totally com fort able with the mate rial in these first two chap ters before pro ceed ing.

Chapter 3

Alerts, Prompts, and User Feedback

Before you con tinue with this book, make sure that you are thor oughly famil iar with the mate rial in Chap ters 1 and 2. That mate rial pro vides the fun da men tal build ing blocks that I will use through out the rest of the book to guide you through build ing a com plete web site. With out a basic grasp of HTML and the fun da men tals of JavaScript, it will be dif fi cult for you to work through the rest of this book.

For edu ca tional pur poses I am going to walk you through the pro cess of building a com plete web site for a fic ti tious busi ness. Since I hap pen to enjoy antique book col lect ing, we are going to build a web site for an antique bookstore. How ever, the tech niques are com mon to all busi nesses. Each chap ter will first dis cuss a par tic u lar tech nique or tech niques and then give you some exam ples using those tech niques. Then the chap ter will use one or more of them dem on strated in our ficticious busi ness web site we are build ing.

Alert Boxes

One of the sim plest things to do with JavaScript is the alert box. An alert box is sim ply a small mes sage box that pops up and gives the user some infor mation. Let's start by add ing a sim ple greet ing to vis i tors. This greet ing will be jazzed up as we go along, with more inter est ing fea tures.

Con sider the fol low ing exam ple:

Example 3-1

```
<HTML>
<HEAD>
    <TITLE>Alert Box</TITLE>
```

```
<SCRIPT LANGUAGE="JavaScript">
    function alertMe(message)
    {
      alert(message)
    }
alertMe("Welcome to my web page!")
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

This may seem like a rather triv ial item to add to a web page, but you should note two things. The first is that some sim ple pop-up boxes and dynamic content can eas ily dif fer en ti ate your web site from oth ers. The sec ond thing you should keep in mind is that this script shows you how to use alert boxes. An alert box is a built-in JavaScript func tion that allows you to dis play mes sages to the user. All you have to pass it is either a vari able or a lit eral string value you want to dis play. You will see, through out this book, that this is a very useful tech nique and will be used fre quently.

If you entered the code cor rectly for Exam ple 3-1 you should see this image:

Fig ure 3-1

Prompt Boxes

Now we are going to expand this script to use a prompt box, and to have dif fer-ent action depend ing on the user's response. A prompt box is very sim i lar to an alert box, only it allows the user to enter in data. The fol low ing script gives you a basic idea of how a prompt box works.

Example 3-2

```
<HTML>
<HEAD>
    <TITLE>Alert Box</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!–
var name = prompt("Please Enter your name:", "John Doe")
alert("Hello " + name)
// –>
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

This script begins by using a prompt box to inquire as to the user's name. The value of the user's input is placed into a vari able called name using the assign-ment oper a tor = (remem ber that a sin gle = assigns a value to a vari able whereas a = = eval u ates whether or not a vari able is equal to a par tic u lar value).

```
var name = prompt("Please Enter your name:", "John Doe")
```

Please note that we do pro vide the user with a default value, in case he or she does not enter any value. Then we sim ply dis play a greet ing to the web site vis i tor using the name he or she just pro vided. The fol low ing fig ures show these alert boxes.

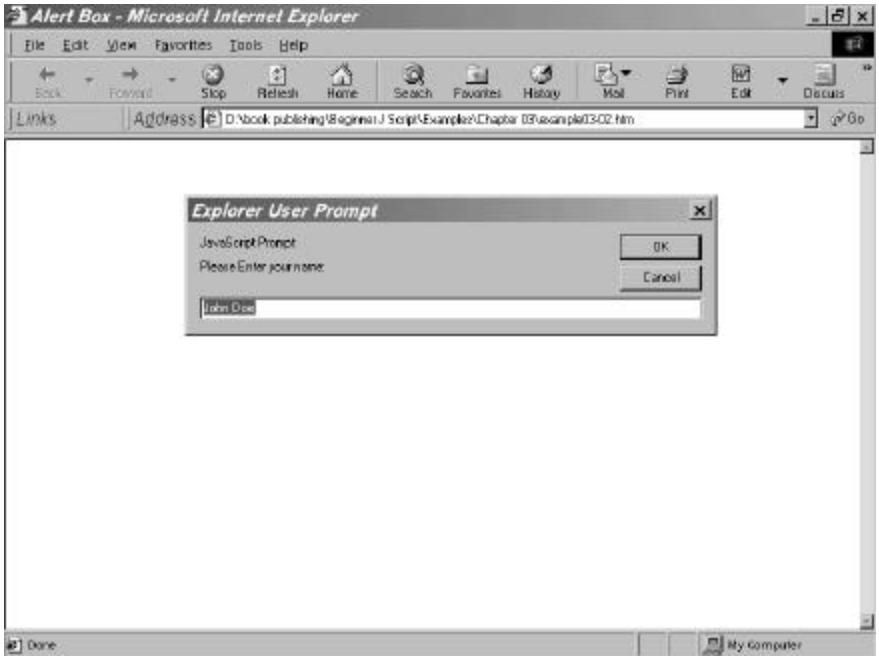


Figure 3-2

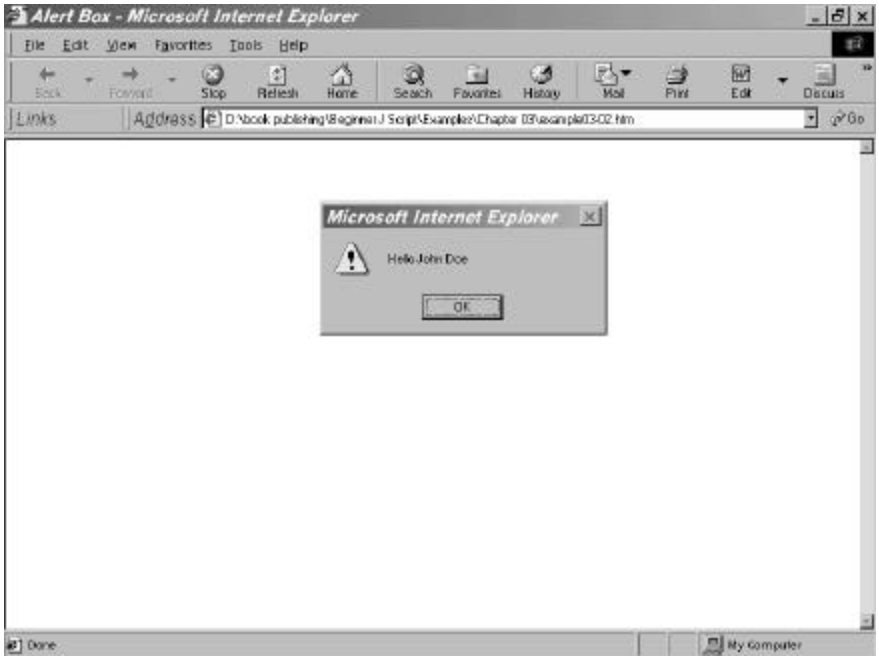


Figure 3-3

The next exam ple dem on strates the use of a prompt box in con junc tion with an alert box and an if state ment. This script asks the user what browser he or she is using and responds accord ingly.

The next thing we ask the user to do is to indi cate whether or not he or she is using a recent ver sion of their browser. You should notice, by this point, that all input boxes have OK and Can cel but tons. If the user clicks Can cel, the variable that is assigned the value of the prompt will con tain a null value. Also notice that the only struc tural dif fer ence between the two prompt boxes is the use of the word prompt prior to the paren the sis.

```
var name = prompt("Please Enter your name:", "John Doe")
var message = "Click OK if you are using Netscape 3.0 or above, or Internet
Explorer 4.0 or above"
```

If you place the word prompt before the open ing paren the sis, JavaScript will cre ate a text field in the prompt box it dis plays, so that the user can enter data. With out the prompt com mand, your prompt box will only allow the user to choose between OK and Can cel.

The sec ond ques tion is to make sure the user has a recent ver sion of their browser. Older brows ers do not sup port all of the JavaScript com mands we might use in the cre ation of our web site. After the user tells us whether or not he or she has a recent ver sion we use an if state ment to choose our next course of action.

```
if (!confirm(message))
{
alert("Please download the latest version of Netscape or Internet Explorer.")
}
else
{
        alert ("Hello " + name )
}
```

The fol low ing sim ple script illus trates for you the use of alert boxes and two dif fer ent types of prompt boxes, com bined with an if-else code seg ment to create a use ful addi tion to our web site.

Example 3-3

```
<HTML>
<HEAD>
   <TITLE>Prompt box 2</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!—
```

```
var name = prompt("Please Enter your name:", "John Doe")
var message = "Click OK if you are using Netscape 5.0 or above, or Internet
Explorer 5.0 or above"
if (!confirm(message))
{
   alert("Please download the latest version of Netscape or Internet Explorer.")
}
else
{
        alert ("Hello " + name )
}
// –>
</SCRIPT>
</HEAD>
<BODY>

</BODY>
</HTML>
```

Writing Back to the Web Page

Alert boxes and prompt boxes are fairly easy meth ods for pro vid ing some level of user inter ac tion. You can also use JavaScript to write infor ma tion directly back onto the web page. Essen tially, an alert box is per fect if you wish to dis-play some thing and then have it dis ap pear as soon as the user acknowl edges it. How ever, if you want infor ma tion to stay on the screen the entire time the user is on your site, then you need to write that to the actual HTML document.

If we replace the pre vi ous script with the fol low ing one we will be able to retain the cur rent user's name on the page for as long as they are vis it ing the site.

Example 3-4

```
<HTML>
<HEAD>
   <TITLE>Writing to the web page</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var name = prompt("Please Enter your name:", "John Doe")
var message = "Click OK if you are using Netscape 3.0 or above, or Internet
Explorer 4.0 or above"
if (!confirm(message))
{
```

```
   alert("Please download the latest version of Netscape or Internet Explorer.")
}
else
{
        document.write(name)
}
// -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR= White>
</BODY>
</HTML>
```

Note that the only real dif fer ence is this seg ment:

```
else
{
        document.write(name)
}
```


Instead of putt ing up an alert box, we write the user's name onto the actual web page. Notice the use of the doc u ment object. This object rep re sents the web page that is cur rently loaded into the browser. You can do a lot of inter est-ing things with this object, as you will see. Now writ ing a line to the HTML page is mod er ately inter est ing. But what is more inter est ing is the fact that you can use the doc u ment.write() method to actu ally write new HTML into the doc u ment. Take a look at this exam ple:

Example 3-5

```
<HTML>
<HEAD>
   <TITLE>Writing to the web page</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var name = prompt("Please Enter M if you are male and F if you are female", "M")
if(name=='M')
{
        document.write('<FONT COLOR = Blue>')
        document.write('<H1>Yo Dude!</H1>')
        document.write('</FONT>')

}
else
{
        document.write('<FONT COLOR =Pink>')
        document.write('<H1>You go girl!</H1>')
        document.write('</FONT>')
```

```
}
</SCRIPT>
</HEAD>
</BODY>
</HTML>
```

If you entered the code prop erly you should see a series of images like these:

Figure 3-4

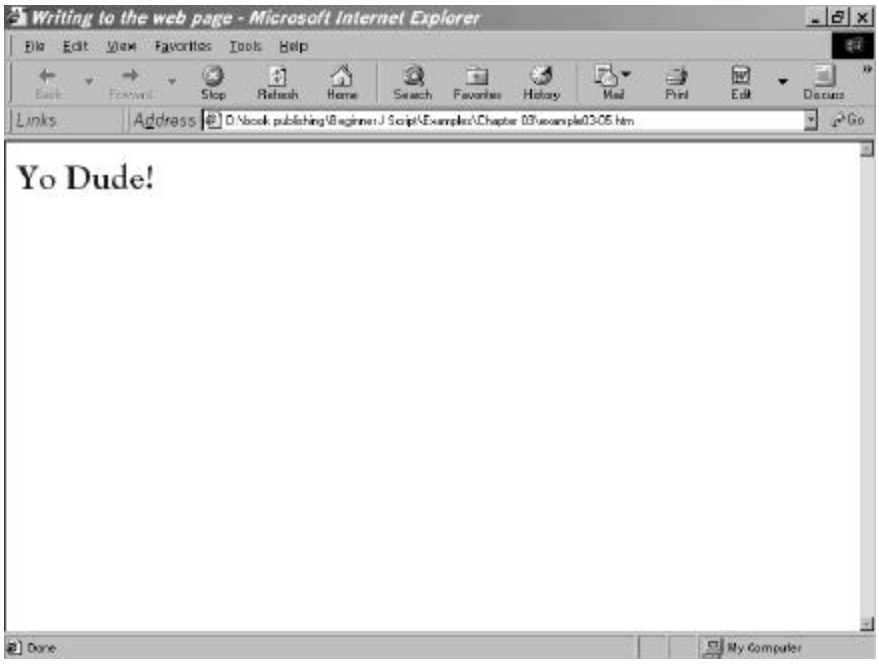


Figure 3-5

In this exam ple we take the input of the user from a prompt box and then we type actual HTML code into the exist ing web page. You can use the doc u-ment.write() method to type any valid HTML code. This allows you to alter the actual HTML con tent of your page based on the user's input. Now I want you to take just a moment to con sider this. You can eas ily cus tom ize your web page based on a user's input. Again, let me stress that you can write any valid HTML code with this method. It may be a very sim ple tech nique, but in my opin ion it's one of the cool est tech niques.

Antique Bookstore Project

Now we need to find a way to incor po rate this script into our ongo ing antique book store page. The fol low ing code pro vides the source for the antique bookstore page (at least the begin nings of it) and incor po rates alerts and prompt boxes.

Example 3-6

```
<HTML>
<HEAD>
   <TITLE>Ye Olde Book Shoppe</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!—
var name = prompt("Please Enter your name:", "John Doe")
var message = "Click OK if you are using Netscape 5.0 or above, or Internet
Explorer 5.0 or above"
if (!confirm(message))
{
        alert("Please download the latest version of Netscape or Internet
Explorer.")
}
else
{
        alert ("Hello " + name )
}
// —>
</SCRIPT>
</HEAD>
<BODY BGCOLOR= White>
<CENTER>
<P><TABLE BORDER=1 CELLSPACING=0 CELLPADDING=0>
   <TR>
      <TD>
         <P><IMG SRC="gargshelf.gif" >
      <TD>
         <P><CENTER>Ye Olde Book Shoppe</CENTER>
    <TD>
         <P><IMG SRC="gargshelf.gif" >
</TABLE>
</CENTER>
<P>
</BODY>
</HTML>
```

We will put this into the main.htm page, thus giv ing a greet ing when the user first loads the page.

Summary

This chap ter showed you how to give the user feed back using alerts and how to get infor ma tion from the user using prompt boxes. It also showed you how to write directly to the HTML doc u ment. These sim ple tech niques pro vide the ground work for your expand ing knowl edge of JavaScript. These scripts also pro vide some basic user inter ac tion with your web site. Pay par tic u lar attention to the doc u ment object, as you will see it in many places through out this book.

Chapter 4

Image Effects

One of the most use ful aspects of JavaScript is that it makes it quite easy for you to work with images. You can do a num ber of fas ci nat ing things with images in JavaScript. Some of the image tech niques we will dis cuss in this chap ter are merely fas ci nat ing "eye candy" to add to your website. Oth ers have a much more prac ti cal value.

The Image Object

JavaScript has a built in object called Image. We will be using this object in most of the scripts in this chap ter. The Image object allows you to cre ate objects that rep re sent images of any type you could place in an HTML page (.JPG, .GIF, .BMP) . To cre ate an image object you sim ply cre ate a vari able as the Image type, like this:

```
Var myimage = new Image()
```

The Image object has some prop er ties you will need to use. The most important is the src prop erty. This des ig nates the actual image file you will use as a source for your Image object. The syn tax for using this prop erty is:

```
imageName.src = "myimage.gif"
```

The Image object can only be used to work with images already placed into the HTML page. You can not add new images or com pletely delete images. You can, how ever, replace images with other images. This tech nique is best illustrated in what is com monly called a roll over.

Rollover

The roll over is one of the most com monly used JavaScript imag ing tech niques. The essen tial pur pose of the roll over is quite sim ple. When you use an image as an link, you can use the roll over method to cause that image to change when your mouse moves over it.

Let's assume your web page has an image on it that is used as a link to another page. What we want to accom plish is to have that image change when the user moves the mouse over the image. Below is an exam ple of how to do this:

```
if(document.images)
{
     image1 = new Image
    image2 = new Image
    image1.src = "mysecondimage.gif"
    image2.src = "myfirstimage.gif"
}
else
{
    image1=""
    image2= ""
}
</SCRIPT>
```

Below is the code you put in your HTML body wher ever you wish the link/roll over to appear.

Example 4-1

```
<a href ="nextpage.htm"
    onMouseover = "document.pic.src =image1.src"
    onMouseout = "document.pic.src =image2.src">

<img src = "myfirstimage.gif" name = "pic">

</A>
```

If you entered all the code prop erly you should observe some thing like this:

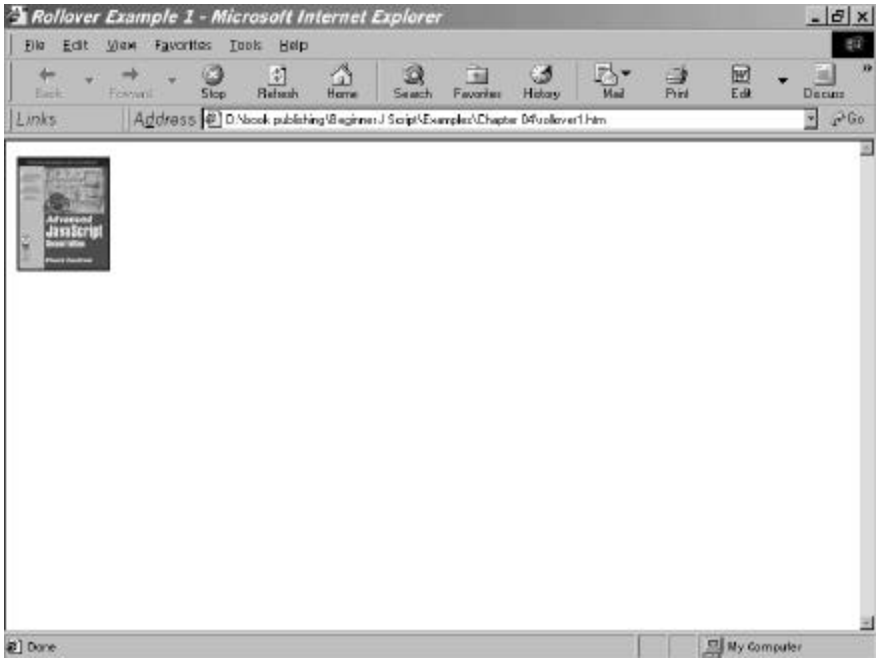


Fig ure 4-1

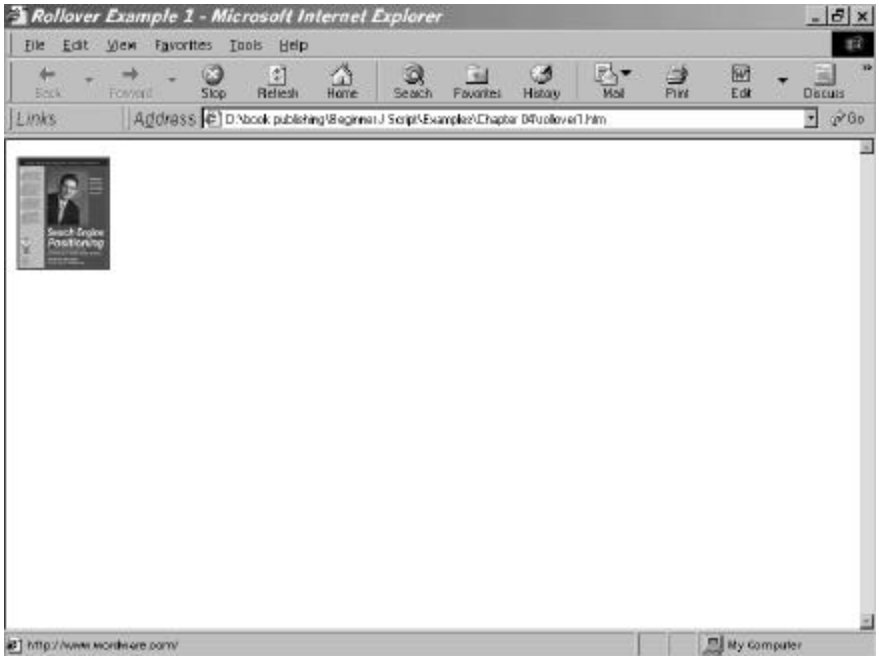


Fig ure 4-2

Let’s exam ine this code for a moment. What the JavaScript is doing is cre at ing vari ables that are instances of the Image object. Then we are set ting the src

prop erty (the source) for those image objects, equal to some pic ture on your hard drive. When the web page first loads, the link will ini tially dislplay "myfirstimage.gif". The space where that image is located, we have named 'pic'. When the mouse moves over the image/link we change the image space named pic, to con tain the image in image1. When we move the mouse out of the image, we change it back to the pic ture in image2. The 'doc u ment' key-word is a new key word for you. It refers to the cur rent HTML doc u ment that is loaded in the browser. This key word will become even more impor tant in later chap ters.

Slide Show

JavaScript allows you to cre ate a fas ci nat ing slide show. Basically a sin gle image is ini tially dis played and the user can scroll back and forth between dif-fer ent images. This is an out stand ing fea ture to add to any e-commerce site. It allows your website vis i tors to view all of your prod ucts, with out clut ter ing the web page with doz ens of images at one time. Let's look at a sam ple of how to do this:

Example 4-2

```
<HTML>
<HEAD>
  <TITLE>Slideshow Sample</TITLE>
  <SCRIPT LANGUAGE="JAVASCRIPT" >
       PicArray = new Array("pic1.jpg","pic2.jpg","pic3.jpg")
       CurrentPic = 0
       ImageCount = PicArray.length - 1
       function MovePrevious()
       {
             if (document.images && CurrentPic > 0)
             {
                        CurrentPic–
                        document.myPicture.src=PicArray[CurrentPic]
              }
       }
     function MoveNext()
     {
              if (document.images && CurrentPic < ImageCount )
              {
                        CurrentPic++
                        document.myPicture.src=PicArray[CurrentPic]
              }
```

```
}

</SCRIPT>
</HEAD>
<BODY BGCOLOR="WHITE">
<CENTER>
<H1>My Slide Show</H1>
<IMG SRC="pic1.jpg" NAME="myPicture" ALT="My First Slideshow">
<BR>

<A HREF="javascript:MovePrevious()">&lt;&lt;  Move Previous</A>
<A HREF="javascript:MoveNext()">Move Next &gt;&gt;</A>
</CENTER>
</BODY>
</HTML>
```

Let's take a closer look at this script and see what is going on here. To begin with, in our JavaScript we are defin ing some vari ables:

```
 PicArray = new Array("pic1.jpg","pic2.jpg","pic3.jpg")
CurrentPic = 0
ImageCount = PicArray.length - 1
```

The first vari able, PicArray, is sim ply an array of images. The sec ond is a number indi cat ing what ele ment in the array we are cur rently look ing at. All arrays start with zero so we ini tial ize CurrentPic to 0. ImageCount is sim ply a number tell ing us how many images we cur rently have.

The next thing we do in this par tic u lar script is to define two func tions, MovePrevious() and MoveNext(). The pur pose of these func tions is fairly obvi ous, to move for ward or back ward through the slide show. In each of these func tions we have an if state ment that sim ply asks two ques tions: Does this web page/browser sup port the image object (if doc u ment. Images) and are we already at the limit of our slide show (if you are at the begin ning, you can not very well move pre vi ous)? If both of these con di tions are met, then exe cute the func tion.

NOTE: The symbol && denotes a logical AND, that is, are both values true. The || is a logical OR, that is, is one of the values true.

The MovePrevious func tion sim ply uses the dec re ment oper a tor (—) to reduce CurrentPic and then dis plays that par tic u lar image from the image array PicArray. The MoveNext func tion is iden ti cal except it uses the increment oper a tor (+ +).

The code in the body of the HTML doc u ment is a stan dard HTML ref er ence, but rather than being a ref er ence to another web page or an email address, it is a ref er ence to a func tion in our JavaScript.

If you enter the code prop erly, you can view a series of images like the ones shown here.

Fig ure 4-3

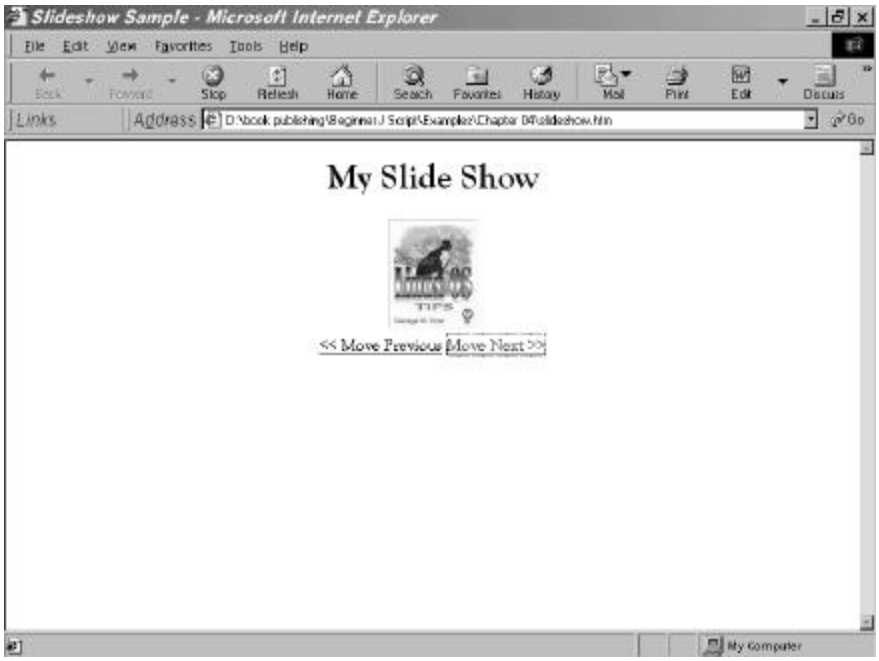


Fig ure 4-4

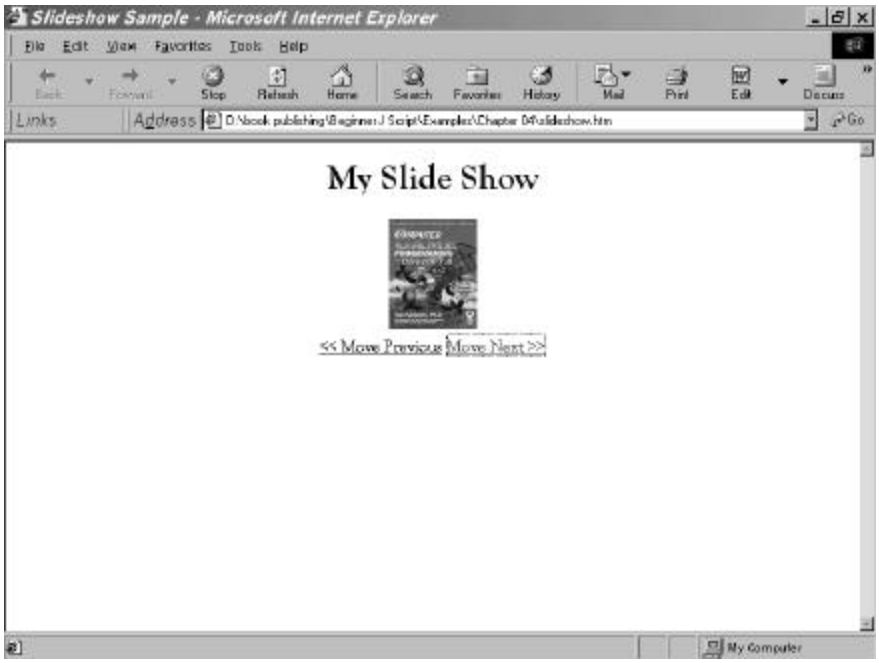


Fig ure 4-5

A slide show is a very prac ti cal piece of JavaScript that you will be able to use to enhance any site. How ever, it is ide ally suited to pre sent ing prod ucts for sale.

In our first slide show exam ple we used links to call the JavaScript func tions. You can also use but tons to cre ate the same effect. All you need to do is replace the <A HREF> link sec tion with code to place but tons on the screen. If you wish to do this sim ply replace:

```
<A HREF="javascript:MovePrevious()">&lt;&lt; Move Previous</A>
<A HREF="javascript:MoveNext()">Move Next &gt;&gt;</A>
```

With:

```
<FORM>
     <INPUT TYPE="button" VALUE="<–" onClick="MovePrevious()">
     <INPUT TYPE ="button" VALUE ="–>" onClick="MoveNext()">
</FORM>
```

If you do this prop erly, it will pro duce the image you see here.

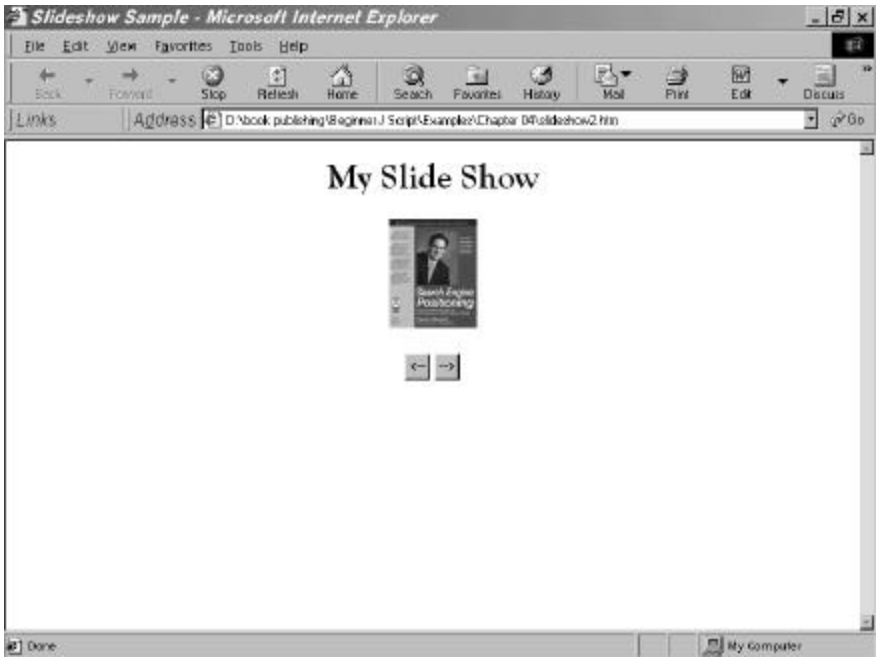


Fig ure 4-6

In case the HTML <FORM> tag is new to you, let's exam ine it. The <FORM> and </FORM> tags sim ply denote that what lies between them will be form ele ments. This is just like any of the other tags you have used

before. INPUT TYPE tells the browser what type of form ele ment to place on the web page. VALUE sim ply places a cap tion on the but ton you have cre ated. OnClick = sim ply says that when the but ton is clicked to do what ever comes after the equals sign. In our case it is a sim ple call to one of our JavaScript functions.

You could also use the <A HREF> link method we orig i nally used, and instead of using words such as “move pre vi ous” you can use images. You sim-ply replace the <A HREF> link sec tion in the first slide show exam ple with this:

```
<A HREF="javascript:MovePrevious()"><IMG SRC ="left.gif"</A>
<A HREF="javascript:MoveNext()"><IMG SRC ="right.gif"</A>
```

If you do this prop erly, it will gen er ate an image like this:

Fig ure 4-7

Image Preview

Have you ever vis ited a web site that you felt was sim ply clut tered with too many images? I cer tainly have. One way to com bat this is the slide show we just did. Another method is to ini tially dis play the images as small images, and allow the user to select which one they wish to view in full size. Here is a piece of code that will let you do just that.

Example 4-3

```
<HTML>
<HEAD>
   <TITLE> Image Preview
  </TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT">
if (document.images)
{
   text1 = new Image
   text2 = new Image
   text3 = new Image
   text4 = new Image

   text1.src = "pic1.jpg"
   text2.src = "pic2.jpg"
   text3.src = "pic3.jpg"
   text4.src = "email2.gif"
}
else
 {
   text1= ""
   text2 = ""
   text3 = ""
   text4 = ""
   document.textField = ""
}
</SCRIPT>
</HEAD>
<BODY BGCOLOR=white>
<TABLE>
    <TD>
    <A HREF="page1.htm" onMouseover="document.textField.src=text1.src">
         <IMG SRC="pic1.jpg"WIDTH = 25 HEIGHT = 25 ></A>
    <BR>
<A HREF="page2.htm" onMouseover="document.textField.src=text2.src">
         <IMG SRC="pic2.jpg"WIDTH = 25 HEIGHT = 25></A>
         <BR>
<A HREF="page3.htm" onMouseover="document.textField.src=text3.src">
         <IMG SRC="pic3.jpg"WIDTH = 25 HEIGHT = 25 ></A>
<TD>
         <IMG SRC="doors.gif"  NAME="textField" ALT="Text Field">
</TABLE>
</BODY>
</HTML>
```

Let us exam ine this script and see what is hap pen ing. The first thing we see is the cre ation of some Image objects. We have seen this in pre vi ous scripts so it is really noth ing new. In fact the only thing really new about this script is that when the images are placed on the web page they are placed with a very small size. Then when the mouseover event is fired, the image is dis played in full size to the right.

Banner Ads

You have likely seen ban ner ads on web pages. It's actu ally pretty sim ple to do these in JavaScript. The fol low ing script takes three images and rotates through them at a given inter val.

Example 4-4

```
<HTML>
<HEAD>
    <TITLE>Banner</TITLE>
    <SCRIPT LANGUAGE="JAVASCRIPT">
         ImageArray = new Array("banner1.gif","banner2.gif","banner3.gif")
         CurrentImage = 0
         ImageCount = ImageArray.length

        function RotateBanner()
        {
                  if (document.images)
                  {
                            CurrentImage++
            if (CurrentImage ==ImageCount)
                             {
                CurrentImage = 0
                               }
                                 document.Banner.src=ImageArray[CurrentImage]
                               setTimeout("RotateBanner()", 5000)
                     }
          }

</SCRIPT>
</HEAD>
<BODY BGCOLOR="WHITE" onLoad="RotateBanner()">
<CENTER>
        <IMG SRC="banner1.gif"  NAME="Banner" ALT=" Banner">
</CENTER>
</BODY>
</HTML>
```

If you exam ine this script closely you will see that it is using tech niques we have already seen but in a new way. So let's take a look at what is new in this script. To begin with in the <BODY> tag you now see onLoad=RotateBanner(). This sim ply says to start our script when the page is ini tially loaded start our script. You will also notice this line in the script:

```
setTimeout("RotateBanner()", 5000)
```

This is using a built-in JavaScript func tion called Time out. The num ber we pass it is in mil li sec onds. So we are tell ing it to time out every five sec onds. This causes it to load the next image every five sec onds. You can play with the tim ing a bit. How ever, be care ful: If the time is too short, no one will be able to actu ally read the ban ner ad.

Image Popup

Another inter est ing tech nique you can use is to have an image pop up in a new win dow by itself when another image is clicked. Usually you will have a small ver sion of the image, and when the user clicks on it they get the full image in a new win dow.

Example 4-5

```
<HTML>
<HEAD>
<TITLE>Example 04-05</TITLE>
<SCRIPT LANGUAGE="JavaScript" >
function newWindow(imagename)
{
   imageWindow = window.open(imagename, "imageWindow", "width=320,height=240)
}
</SCRIPT>
</HEAD>
<BODY BGCOLOR="WHITE">
<H4>Double click on one of the following images to see the full size image</H4>
<IMG SRC="pic1.jpg" HEIGHT = 50 WIDTH = 50 onDblclick="newWindow('pic1.jpg')">

<IMG SRC="pic2.jpg" HEIGHT = 50 WIDTH = 50 onDblclick="newWindow('pic2.jpg')">

<IMG SRC="pic3.jpg" HEIGHT = 50 WIDTH = 50 onDblclick="newWindow('pic3.jpg')">
</BODY>
</HTML>
```

This code, if prop erly entered will pro duce an image like the fol low ing:

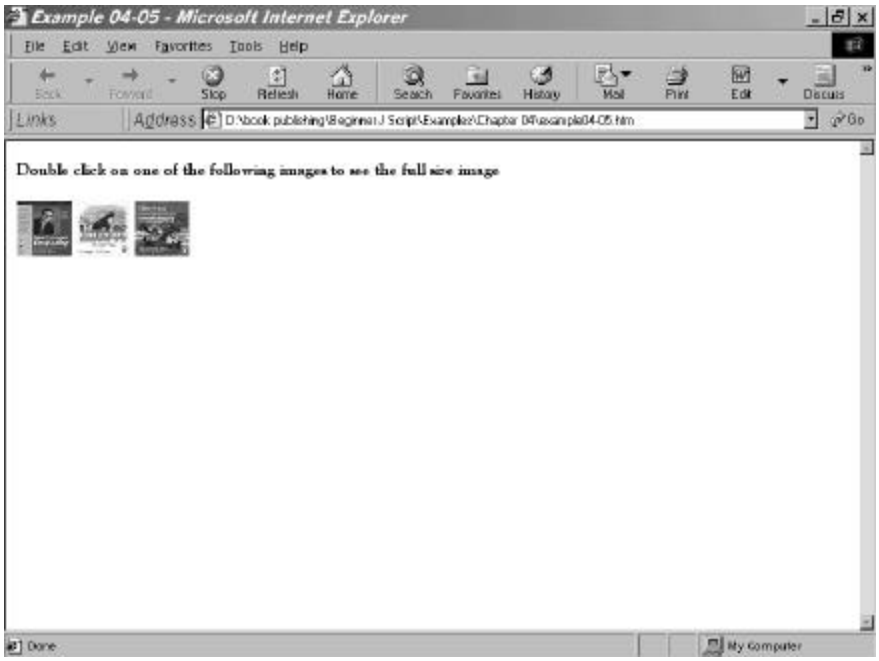


Figure 4-8

If you click on one of the images, it will pop up that image, full size, in a new window, as shown here:

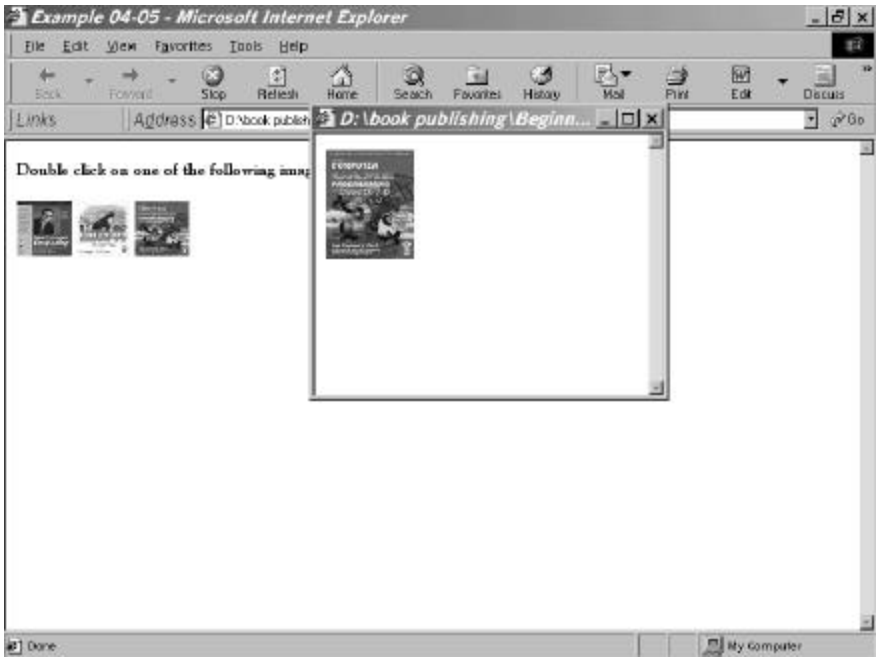


Figure 4-9

Let's take a look at this script and see what is hap pen ing. The actual script por tion is actu ally quite sim ple. The func tion con sists of a sin gle line of code (it does n't get much sim pler than that!):

```
function newWindow(imagename)
{
   imageWindow = window.open(imagename, "imageWindow", "width=320,height=240)
}
```

Sim ply pass any image name to this func tion and it will launch a new win dow show ing that pic ture. The way we open the new win dow is to use the 'open' method of the 'win dow' object. This method takes four required param e ters and sev eral optional ones. The first param e ter tells it what to open. In our exam ple it is an image, but you can open another HTML page as well. The sec ond param e ter gives JavaScript a name to refer to this new win dow by. Notice that this name matches the vari able I chose. The next two param e ters sim ply deter mine the height and width of the win dow. There are many more optional param e ters but they will be dis cussed in Chap ter 6.

Antique Bookstore Project

Now we can add some of these inter est ing tech niques to our antique book-store pro ject. This will add some inter est ing fea tures. First we are going to place a new web page in our pro ject that uses the slide show to demo some books. This page is in the Antique Book store folder on the CD and is named books.htm. What I have done with that page is sim ply add in the slide show script, only I have used but tons to dis play the images.

```
<HTML>
<HEAD>
   <TITLE>Book Inventory</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT" >
   PicArray = new Array("book1.gif","book2.gif","book3.gif","book4.gif")
CurrentPic = 0
ImageCount = PicArray.length - 1
function MovePrevious()
{
      if (document.images && CurrentPic > 0)
      {
          CurrentPic–
          document.myPicture.src=PicArray[CurrentPic]
      }
}
```

```
          function MoveNext()
        {
        if (document.images && CurrentPic < ImageCount )
          {
          CurrentPic++
          document.myPicture.src=PicArray[CurrentPic]
          }
       }
</SCRIPT>
</HEAD>
<BODY BGCOLOR = white >
<P>
<CENTER>
<IMG SRC="book1.gif" HEIGHT=300 WIDTH =300 NAME="myPicture" ALT="Our Book
Inventory">
     <FORM>
          <INPUT TYPE="button" VALUE="<–" onClick="MovePrevious()">
          <INPUT TYPE ="button" VALUE ="–>" onClick="MoveNext()">
     </FORM>
</CENTER>
</BODY>
</HTML>
```

We also need to add a ban ner ad to the main.htm web page. For that I used a ban ner ad exactly like the one used ear lier in this chap ter. If you have been follow ing along with all the exam ples, you should be start ing to see a very inter est ing web site take shape.

Summary

In this chap ter you have learned how to use var i ous image mod i fi ca tion scripts to cre ate a vari ety of inter est ing visual effects. It is impor tant that you actu ally try each of these scripts before mov ing on. The prin ci ples you learn here will be car ried through out the book. You will also find that the inter est ing visual effects are the items most demanded by users.

Chapter 5

Background Effects

Web page backgrounds can be somewhat dull. Even if you use an image as the background, it still is static. With JavaScript you can perform a number of interesting effects with the background. Each of these effects will add a great deal of visual impact to your website.

Document Object

You were introduced to the document object in the last chapter. Recall that the document object represents the web page currently loaded into the browser. Using this object we can make changes to that document. Two properties of the document object that we will explore in this chapter are bgcolor and background. These properties are exactly like the tags in HTML. The bgcolor property sets the background color of the web page, and the background property sets an image as the background.

Changing the Background Color

The first script we will examine simply changes the background color to match the color described in a link that the user passes their mouse over. Let's look at that script now:

Example 5-1

```
<HTML>
<HEAD>
    <TITLE>Color Changer</TITLE>
</HEAD>
 <BODY>
<CENTER>
   <a href="/"onmouseover="document.bgColor='blue'">Blue</a>
   <a href="/"onmouseover="document.bgColor='royalblue'">Royal Blue</a>
```

```
    <a href="/"onmouseover="document.bgColor='green'">Green</a>
    <a href="/"onmouseover="document.bgColor='red'">Red</a><BR>
    <a href="/"onmouseover="document.bgColor='magenta'">Magenta</a>
    <a href="/"onmouseover="document.bgColor='pink'">Pink</a>
    <a href="/"onmouseover="document.bgColor='purple'">Purple</a><BR>
    <a href="/"onmouseover="document.bgColor='Skyblue'">Light Blue</a>
    <a href="/"onmouseover="document.bgColor='yellow'">Yellow</a>
    <a href="/"onmouseover="document.bgColor='brown'">Brown</a>
    <a href="/"onmouseover="document.bgColor='white'">White</a><BR>
    <a href="/"onmouseover="document.bgColor='black'">Black</a>
    <a href="/"onmouseover="document.bgColor='coral'">Coral</a>
    <a href="/"onmouseover="document.bgColor='orange'">Orange</a>
</CENTER>
</BODY>
</HTML>
```

In this example we simply set up a standard link where the word displayed is the name of the color we will use as the background color of the web page. When the page first loads, it will look like Figure 5-1.

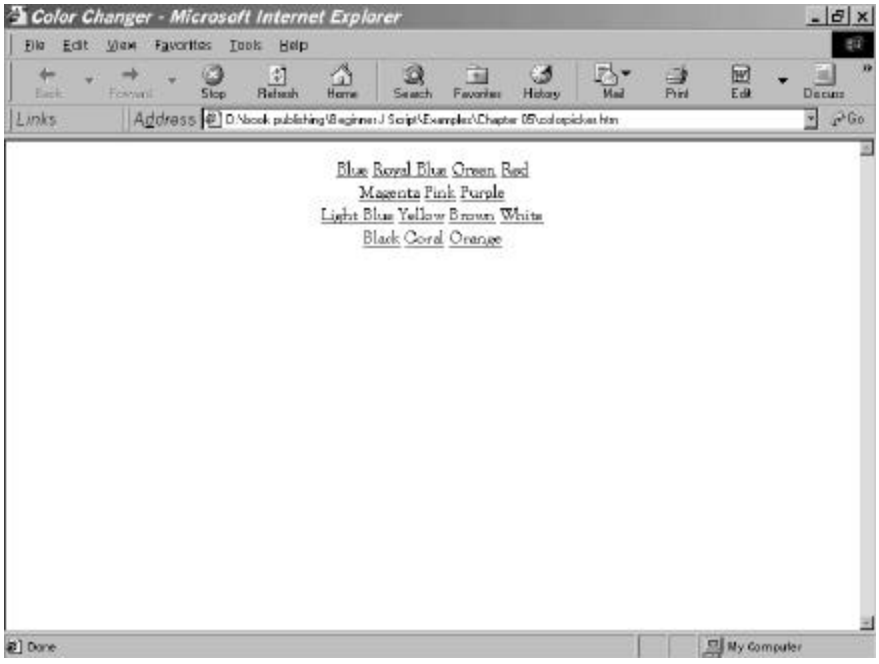


Figure 5-1

If you move your mouse over the word “black,” the image will then look like Figure 5-2:

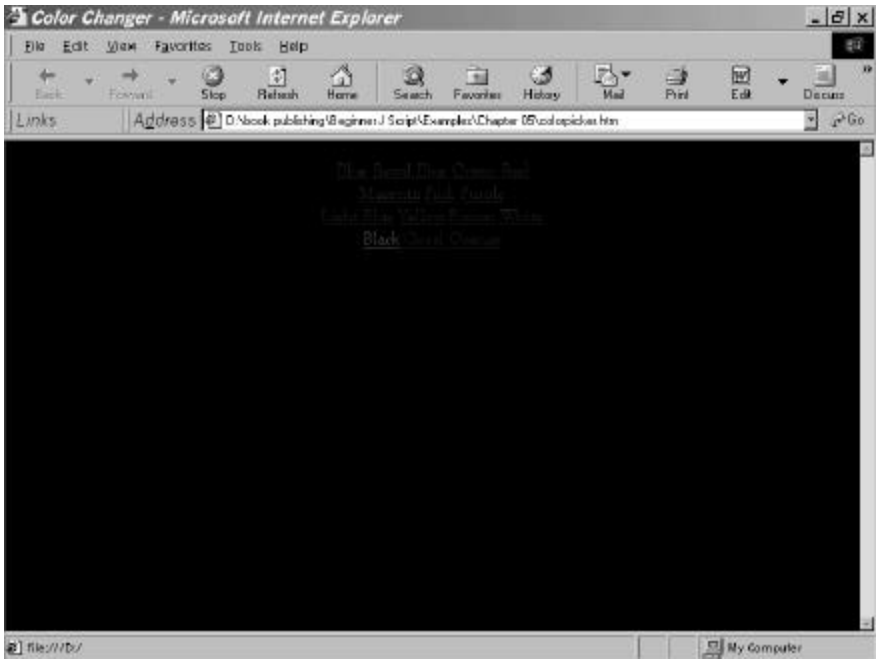


Figure 5-2

And then if you move it over the word “pur ple,” it will look like Fig ure 5-3:

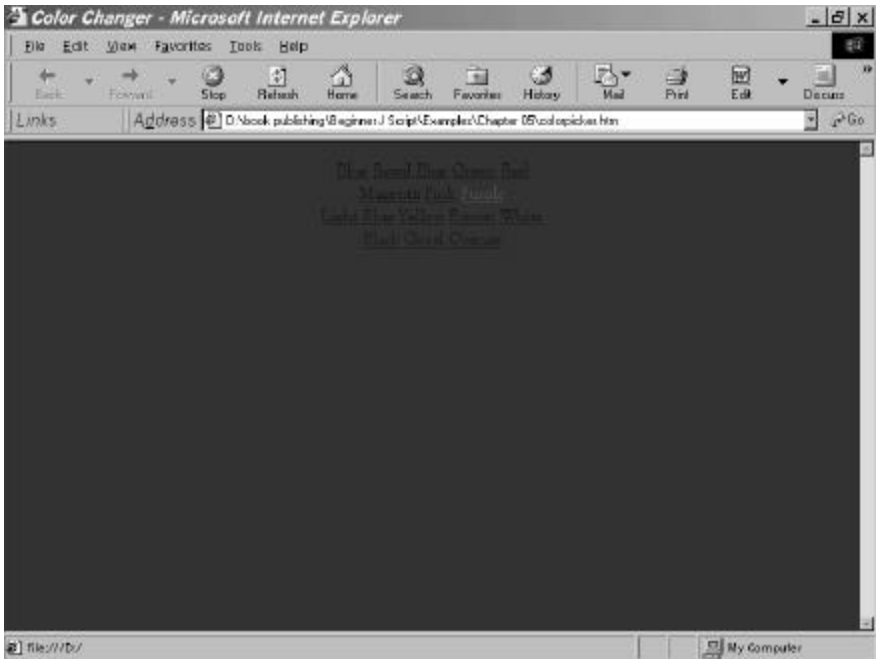


Figure 5-3

Now that you have seen this script in action, let's exam ine what is hap pen ing in the code itself. Each one of the sep a rate links is con structed the same way.

```
<a href="/"onmouseover="document.bgColor='yellow'">Yellow</a>
```

The <a href = por tion should be famil iar to you since it is a stan dard hypertext ref er ence, also known as a link. In this case, how ever, rather than link ing to another web page or to an email address, this sim ply links to itself. So when a mouse moves over the link we have estab lished that the doc u ment object is used to set the back ground color of the web page to the desired color. This script is rel a tively straight for ward but it's also an excel lent way to begin to see the use ful ness of the doc u ment object.

With this par tic u lar script or with vari a tions thereof, you can change the color of the web page doc u ment any time you wish. Some vari a tions to con sider would be to change the back ground color based on the day of the week, time of day, or any other cri te ria you might wish. The only caveat is that you should ensure that the back ground color you use does not make it dif fi cult to view the text on your website.

Changing the Background Image

Next I will show you how to use a tech nique very sim i lar to the one used in Exam ple 5-1 to change the image used in the back ground of your web page, rather than sim ply chang ing the color.

Example 5-2

```
<HTML>
<HEAD>
<TITLE>Image Changer</TITLE>
<SCRIPT LANGUAGE="JavaScript">
   if (navigator.appName == "Microsoft Internet Explorer")
   {
      image1 = new Image()
      image2 = new Image()
      image3 = new Image()
      image4 = new Image()

     image1.src = "back1.gif"
     image2.src = "back2.gif"
     image3.src = "back3.jpg"
     image4.src = "back4.jpg"
   }
function changepicture(imgname)
```

```
    {
    if (navigator.appName == "Microsoft Internet Explorer")
    {
         document.body.background = eval(imgname + ".src");
    }
}

</SCRIPT>
</HEAD>
<BODY>
<CENTER>
[<a href="#" onMouseOver="changepicture('image1');">Marble</a>]
[<a href="#" onMouseOver="changepicture('image2');">Blue Marble</a>]
[<a href="#" onMouseOver="changepicture('image3');">Slate</a>]
[<a href="#" onMouseOver="changepicture('image4');">Dark Marble</a>]
</CENTER>
</BODY>
</HTML>
```

If you enter this code as you see it (and use the images found on the cd), you will get an image such as the one in Fig ure 5-4:

Figure 5-4

Then moving your mouse over the Marble link will give you an image like the one in Figure 5-5:

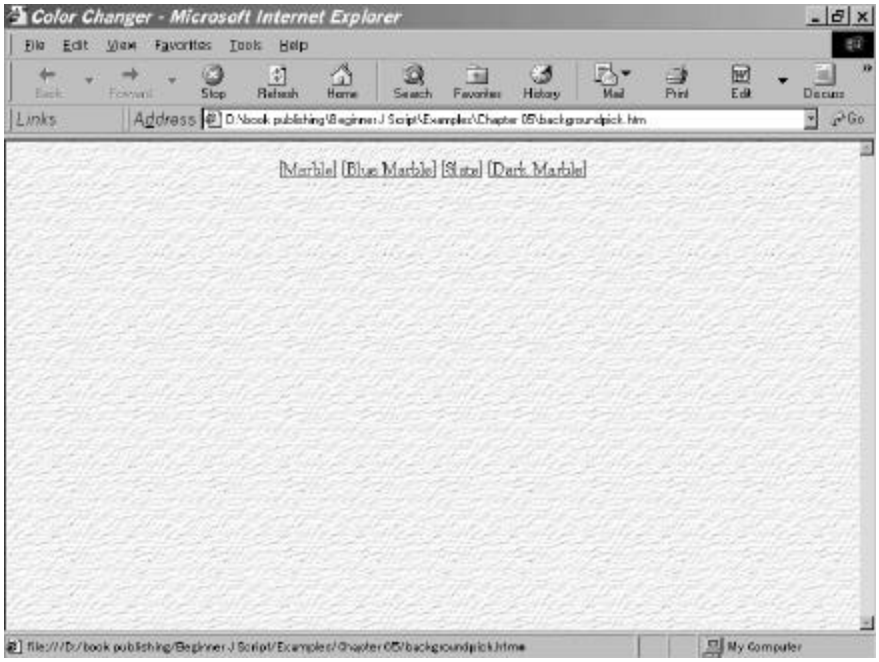


Figure 5-5

Moving the mouse over the Blue Marble link will show you an image such as the one in Figure 5-6:

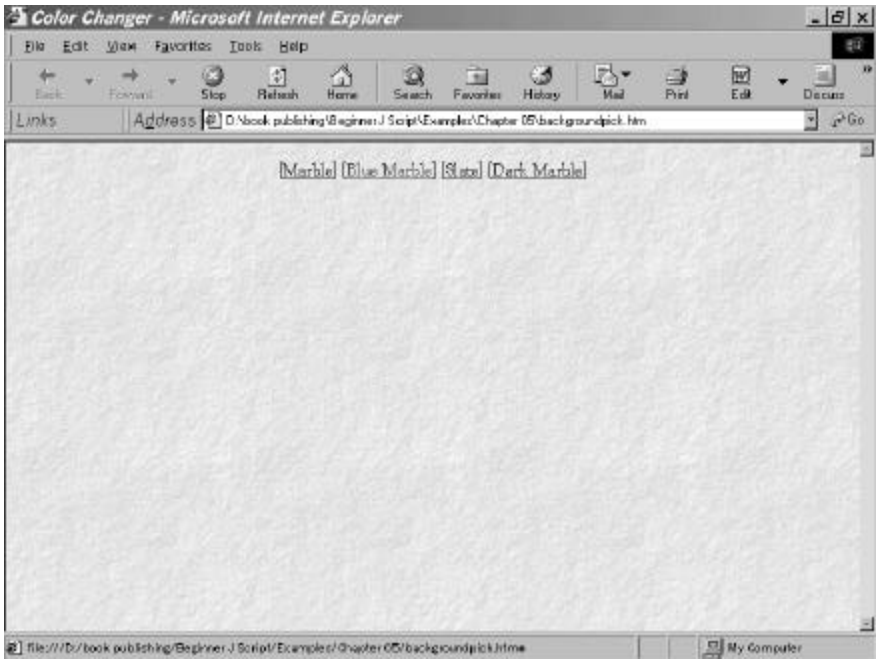


Figure 5-6

Let's take a moment to review the actual code here and make sure that you understand what is happening.

Note that the first line of the script finds out if the browser being used is Internet Explorer. This particular script will not work quite the same in Netscape. We will see in Chapter 6 that detecting the browser and working with it is very important.

```
if (navigator.appName == "Microsoft Internet Explorer")
```

This also introduces you to the Navigator object, which allows you to determine a lot about the browser the person visiting your web site is using. Now assuming that it is Internet Explorer, we then proceed to create instances of the image object and set them to particular images we wish to use for our backgrounds. This code is virtually identical to the rollover code you saw in Chapter 4.

Let's look at another method for changing the background image. In the following example, when the user passes the mouse over a particular image that image becomes the background image for the web page.

Example 5-3

```
<HTML>
<HEAD>
    <TITLE>Background Changer</TITLE>
 <HEAD>
<SCRIPT LANGUAGE="JavaScript">
if (navigator.appName == "Microsoft Internet Explorer")
{
    image1 = new Image()
    image2 = new Image()
    image3 = new Image()
    image4 = new Image()

    image1.src = "back1.gif"
    image2.src="back2.gif"
    image3.src = "back3.jpg"
    image4.src = "back4.jpg"
}
function changepicture(imgname)
{
if (navigator.appName == "Microsoft Internet Explorer")
   {
        document.body.background = eval(imgname + ".src");
   }
}
</SCRIPT>
</HEAD>
<BODY BACKGROUND="back1.gif">
   <A HREF="#" onMouseOver="changepicture('image1');"><img src="back1.gif"></A>
   <A HREF="#" onMouseOver="changepicture('image2');" ><img src="back2.gif"></A>
   <A HREF="#" onMouseOver="changepicture('image3');" ><img src="back3.jpg"></A>
   <A HREF="#" onMouseOver="changepicture('image4');""><img src="back4.jpg"></A>
</BODY>
</HTML>
```

This script works very much like Example 5-2, the only difference being that rather than using text links to the image, it uses an actual picture. When the user passes their mouse over the image, that image becomes the background image for the page. If you write the code correctly, you will get something like Figure 5-7:

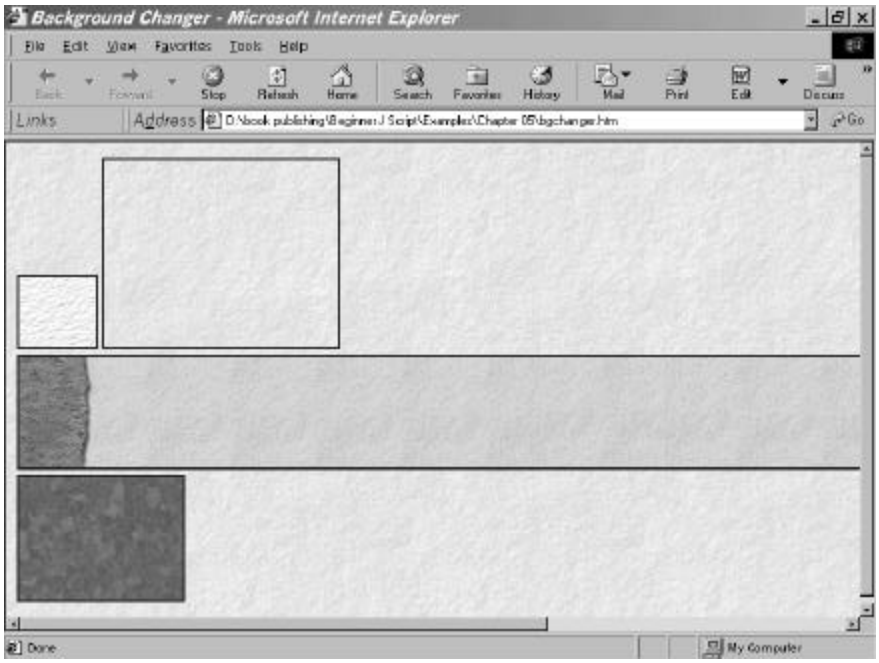


Figure 5-7

Then when you move your mouse over one of the images you will get something like Figure 5-8:

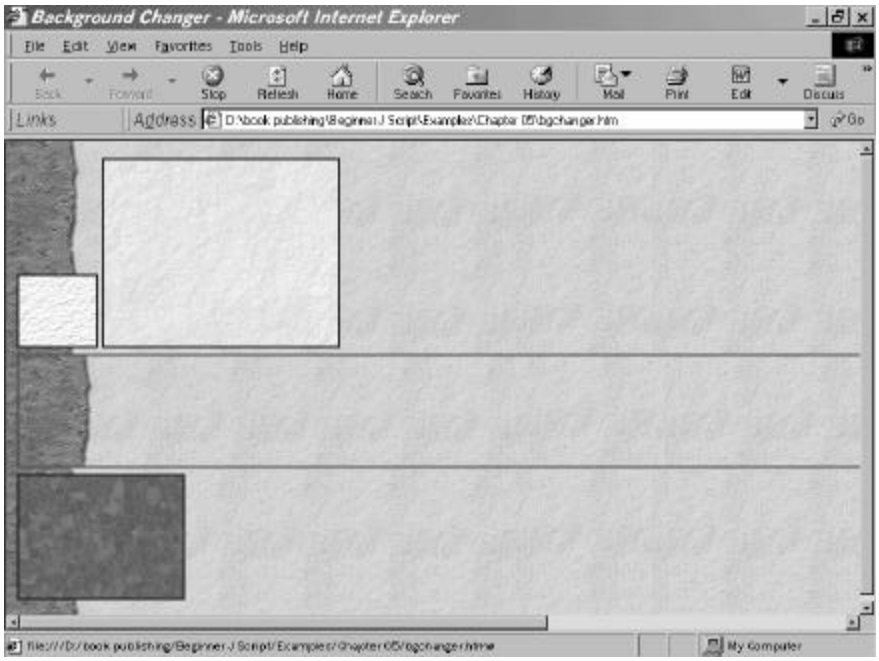


Figure 5-8

This example, much like the previous one (changing the background color) is more important for the principle it demonstrates. It is possible for you to use a different background image based on the date, time, or any other criteria you wish to use. In Chapter 7 "Working with Data and Time" I will show you an example of this.

More Background Effects

In addition to changing the background colors and images, you can use JavaScript to make the background behave in interesting ways. One of the easiest effects to implement is the scrolling background. The following example shows a background that will scroll.

NOTE: This works best with images that are uniform, such as rock textures, stars, lava, etc.

```
<HTML>
<HEAD>
   <TITLE>Background Scroller</TITLE>
</HEAD>

<BODY>
<BODY BACKGROUND="back1.gif">
<SCRIPT LANGUAGE="JavaScript">
    var bgOffset = 0
    var bgObject = eval('document.body')
    function scrollbackground(maxSize)
   {
      bgOffset = bgOffset + 1
      if (bgOffset > maxSize)
               bgOffset = 0

       bgObject.style.backgroundPosition = "0 " + bgOffset;
  }
var scrtimer = window.setInterval("scrollbackground(50)",50)
</SCRIPT>
</BODY>
</HTML>
```

Now let's examine this particular script and see how it works. The first part of the script simply declares two variables:

```
    var bgOffset = 0
    var bgObject = eval('document.body')
```

The first vari able, bgOffset, is sim ply an inte ger rep re sent ing the num ber by which to off set the back ground image's posi tion. The sec ond vari able, bgObject, rep re sents the body of the doc u ment (notice that again we are find-ing uses for the doc u ment object).

Next we have the actual func tion that will do the scroll ing of the back ground:

```
 function scrollbackground(maxSize)
{
   bgOffset++
   if (bgOffset > maxSize)
            bgOffset = 0

    bgObject.style.backgroundPosition = "0 " + bgOffset;
}
```

Some inte ger value is passed to this func tion as maxSize. The first state ment in the func tion sim ply incre ments bgOffset by 1. Then there is an if state ment that checks to see if bgOffset has exceeded maxSize. If so, then bgOffset is reset to zero. We then sim ply set the back ground image posi tion equal to the off set. What this will cause the back ground image to do is to shift down repeat-edly until maxSize is reached, and then go back to the start ing posi tion. The larger the maxSize used, the less fre quently the image will "pop" back up to the top, thus pro vid ing the per son view ing your web page with a more fluid looking background.

Finally we come to the state ment that calls the scrollBackground func tion:

```
var scrtimer = window.setInterval("scrollbackground(50)",50)
```

In this state ment a vari able is declared. That vari able is set equal to the win-dow object's setInterval func tion. Remem ber that the win dow object refers to the browser itself. Inside the setInterval func tion we call the scrollbackground func tion and pass it a max value. What also hap pens here is that a num ber is pro vided that indi cates how fre quently the shift ing will occur. The lower this sec ond num ber is, the faster the image will scroll.

This is per haps, in my view at least, the most inter est ing back ground effect in this chap ter. Pro vided it is used judi ciously it can add a dimen sion to your website not oth er wise obtain able. The pri mary issue to con sider is what image you use. Remem ber that the image is not actu ally scroll ing. We are sim ply cre-at ing the illu sion of scroll ing by shift ing the image pro gres sively fur ther down, then bring ing it back to the top. Because of this an image with a homog e nous

back ground will look like a scroll ing back ground. Images such as stars, water, clouds, stone tex tures, etc. work great for this.

Antique Bookstore Project

Now let's add some of these tech niques to our antique book store pro ject and see how that works. The first thing we are going to include is a scroll ing back-ground on the main page. So let's look at the source code for that page (as it stands so far).

```
<HTML>
<HEAD>
   <TITLE>Ye Olde Book Shoppe</TITLE>
    <SCRIPT LANGUAGE="JAVASCRIPT">
         ImageArray = new Array("banner1.gif","banner2.gif","banner3.gif")
         CurrentImage = 0
         ImageCount = ImageArray.length
        function RotateBanner()
        {
             if (document.images)
             {
                  CurrentImage++
                  if (CurrentImage ==ImageCount)
                  {
                       CurrentImage = 0
                  }
                  document.Banner.src=ImageArray[CurrentImage]
                  setTimeout("RotateBanner()",3000)
             }
        }


</SCRIPT>
</HEAD>
<BODY  background="bod-bg.gif"  onLoad="RotateBanner()">
<CENTER>
<IMG SRC="banner1.gif"  NAME="Banner" >
<P><TABLE BORDER=1>
   <TR>
      <TD>
         <P><IMG SRC="gargshelf.gif">
     <TD>
         <P><CENTER>Ye Olde Book Shoppe</CENTER>
     <TD>
         <IMG SRC="gargshelf.gif">
```

```
</TABLE>
<P>
<P>
</CENTER>
<P>
</BODY>
</HTML>
```

We are going to add a sec ond script to this page. This script goes in the body imme di ately after the < BODY Back ground = > tag:

```
<SCRIPT LANGUAGE="JavaScript">
    var bgOffset = 0;
    var bgObject = eval('document.body');
    function scrollbackground(maxSize)
   {
      bgOffset = bgOffset + 1;
      if (bgOffset > maxSize)
               bgOffset = 0;
       bgObject.style.backgroundPosition = "0 " + bgOffset;
  }
var scrtimer = window.setInterval("scrollbackground(50)",200);
</SCRIPT>
```

The back ground we are cur rently using is not par tic u larly suited for scroll ing so we will change that to a mar ble back ground. This will give us the fol low ing as our main.htm:

```
<HTML>
<HEAD>
   <TITLE>Ye Olde Book Shoppe</TITLE>

    <SCRIPT LANGUAGE="JAVASCRIPT">
        ImageArray = new Array("banner1.gif","banner2.gif","banner3.gif")
        CurrentImage = 0
        ImageCount = ImageArray.length
        function RotateBanner()
        {
             if (document.images)
             {
                  CurrentImage++
                  if (CurrentImage ==ImageCount)
                  {
                       CurrentImage = 0
                  }
                  document.Banner.src=ImageArray[CurrentImage]
                  setTimeout("RotateBanner()",3000)
```

```
            }
        }
</SCRIPT>
</HEAD>
<BODY  background="back1.gif"  onLoad="RotateBanner()">
<SCRIPT LANGUAGE="JavaScript">
    var bgOffset = 0;
    var bgObject = eval('document.body');

    function scrollbackground(maxSize)
   {
      bgOffset = bgOffset + 1;
      if (bgOffset > maxSize)
               bgOffset = 0;

      bgObject.style.backgroundPosition = "0 " + bgOffset;
  }
var scrtimer = window.setInterval("scrollbackground(50)",50);
</SCRIPT>
<CENTER>
<IMG SRC="banner1.gif"  NAME="Banner" >
<P><TABLE BORDER=1>
   <TR>
      <TD>
         <P><IMG SRC="gargshelf.gif">
     <TD>
         <P><CENTER>Ye Olde Book Shoppe</CENTER>
     <TD>
         <IMG SRC="gargshelf.gif">

</TABLE>

<P>
<P>
</CENTER>
<P>
</BODY>
</HTML>
```

This code, if entered properly, should produce the following image:

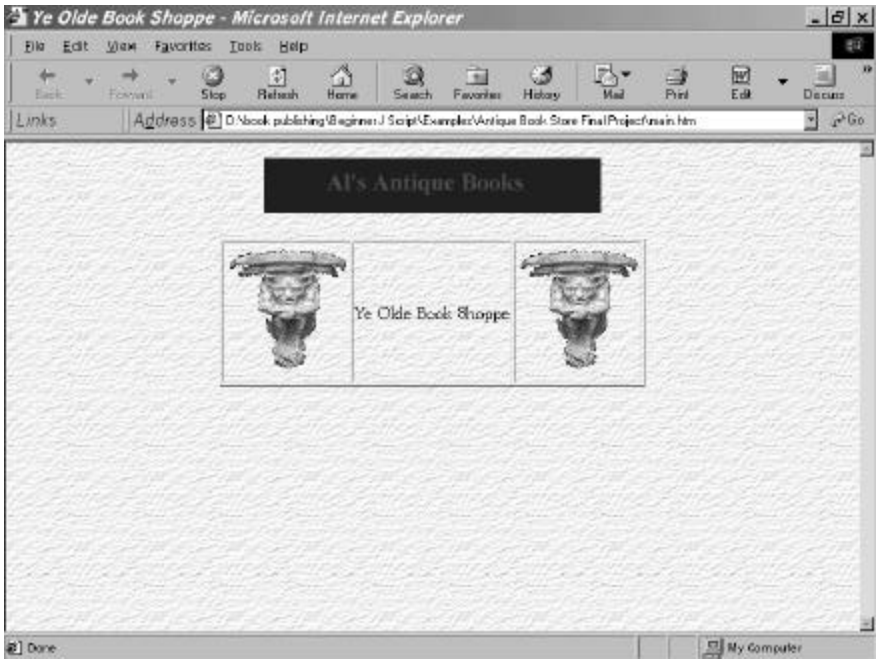


Figure 5-9

Summary

This chapter has introduced you to some simple ways that you can manipulate the background color or image of a web page using JavaScript. Not only have you learned a fairly easy way to "jazz up" your web site, hopefully you have become more comfortable with JavaScript in general. One caution: Do not get carried away with background manipulation. A background that is too busy can be a distraction to site visitors.

Chapter 6

The Document Object Model

With JavaScript you can per form lots of inter ac tion with the browser. In fact, one of the more inter est ing things you can do with JavaScript is to detect the browser being used. You can then use the doc u ment objects to per form dif fer-ent oper a tions. Remem ber from your pre vi ous chap ters that the doc u ment object rep re sents the web page that is cur rently loaded in the browser (i.e., the HTML doc u ment that the JavaScript is run ning in). The Nav i ga tor object rep re sents the browser being used. The win dow object rep re sents the spe cific instance of the browser being used. You can eas ily have mul ti ple instances of the browser open, view ing dif fer ent web pages. We will look at each of these objects in detail in this chap ter and explore ways to uti lize them to enhance your web page.

NOTE: Objects often consist of other objects. The term ***nested objects*** is usually used to characterize such child objects. The nested structure of an object is also known as its ***hierarchy***. The highest-level object is the one that does not have any parent object. Browser objects are classic examples for hierarchy usage, because they are deeply nested. The browser object structure is fairly complex, so it is difficult to remember each and every property at the bottom of the tree. However, I will attempt to familiarize you with the basics of the browser object hierarchy.

window Object

This object is the top-level object of the hier ar chy. It con tains prop er ties that apply to the entire win dow. For exam ple, the sta tus bar of the browser is a prop erty of this object. The win dow object has sev eral prop er ties we will look at.

The first is the .sta tus prop erty. The win dow.sta tus prop erty rep re sents the sta tus bar at the bot tom of your browser. You can use this prop erty to dis play

mes sages to the user in the sta tus bar. We will exam ine some cre ative ways to use this a lit tle later.

The win dow.alert, win dow.con firm, and win dow.prompt prop er ties of the window object allow you to give mes sages to the user and receive feed back.

document Object

By far the most use ful prop erty of the win dow object is the doc u ment object. It con tains prop er ties for the cur rent page loaded in the win dow. With this object you can make all kinds of alter ations to a web page. Almost every thing in the page is a prop erty of the doc u ment object, includ ing links, images, forms and their ele ments, anchors, and more.

The doc u ment object itself has prop er ties you can uti lize. The URL prop erty spec i fies the cur rent URL of the web page. This is a read-only prop erty. A closely related prop erty is the doc u ment.loca tion prop erty. You can use this prop erty to change to another loca tion. That can be quite use ful, and you will see it in sev eral scripts.

The refer rer prop erty is also closely related to the URL and loca tion prop erties. It rep re sents the web page that the user was at just prior to load ing the cur rent page. This can be quite use ful as we will see later on.

There are other prop er ties that give you access to spe cific infor ma tion about the web page you are in. The title prop erty allows you to change the title of the doc u ment. The lastModified prop erty allows you to see when this HTML doc u ment was last mod i fied.

history Object

The his tory object is also a prop erty of the win dow object. It con tains prop erties of the URLs the user has pre vi ously vis ited. This infor ma tion is stored in a his tory list, and is acces si ble through the browser's menu. This object also con tains meth ods enabling you to send the user's browser to a URL found in the his tory list.

The length prop erty sim ply tracks the length of the his tory. It tells you how many items are listed in the his tory. The cur rent prop erty con tains the value of the page you are cur rently in. For nav i ga tion you will use the next and previ ous prop er ties, as they allow you to move for ward and back wards through the his tory.

All of these objects are linked together into a well-defined hier ar chy referred to as the Doc u ment Object Model (DOM). Through cre ative use of the Doc u-ment Object Model, you can do some very inter est ing things in JavaScript. But first you need to become acquainted with this model and what its ele ments do.

image 10-1a (bad file sup plied)

Fig ure 6-1

The Document Object

As pre vi ously noted, the doc u ment object rep re sents the web page that the JavaScript is actu ally run ning in. Tech nically speak ing, the doc u ment object is a prop erty of the win dow object. We have pre vi ously used it to ensure that the web page will sup port the image object. In Chap ter 4 we used if(doc u-ment.images) in our scripts to make sure that the web page would sup port the image object. The doc u ment object has a vari ety of other fas ci nat ing meth ods. How ever, per haps the eas i est, but most ver sa tile, method is the write method.

The write method lit er ally allows you to write to the web page directly. For exam ple, if you sim ply include this line of code:

```
Document.write("Howdy")
```

It will write the word "Howdy" to the web page. Now sim ply writ ing "Howdy" on a web page is not par tic u larly inter est ing. A much more inter est ing appli ca-tion is to dynam i cally change the web page based on some cri te ria you choose.

In the following example, we will ask the web page visitor to identify their gender and alter the appearance of the web page based on their answer.

Example 6-1

```
<HTML>
<HEAD>
    <TITLE>Gender Bender</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var gender = prompt("Enter M for Male and F for female","F")
if(gender=="M")
{
   document.write('<BODY BGCOLOR = Blue>')
   document.write('<FONT COLOR = Red>')
   document.write('Yo Dude')
   document.write('</FONT>')
}
else
{
   document.write('<BODY BGCOLOR = Pink>')
   document.write('<FONT COLOR =Black>')
   document.write('You go girl')
   document.write('</FONT>')
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

This script is really quite simple. It simply asks the user to enter their gender and based on that information it proceeds to alter the properties of the HTML document. You can really expand upon this to have different background images, fonts, and even background music based on the information the user provides you. With our example, if you entered everything correctly you should see a screen like the one shown here:

Figure 6-2

And then you will see one of the fol low ing screens, depend ing on how you answer the prompt shown in Fig ure 6-2:

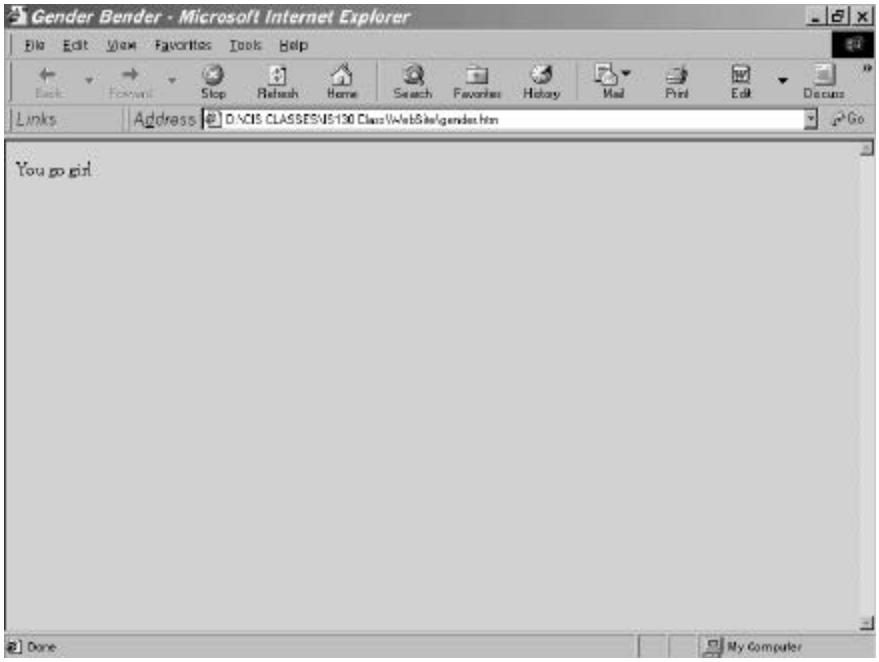


Fig ure 6-3

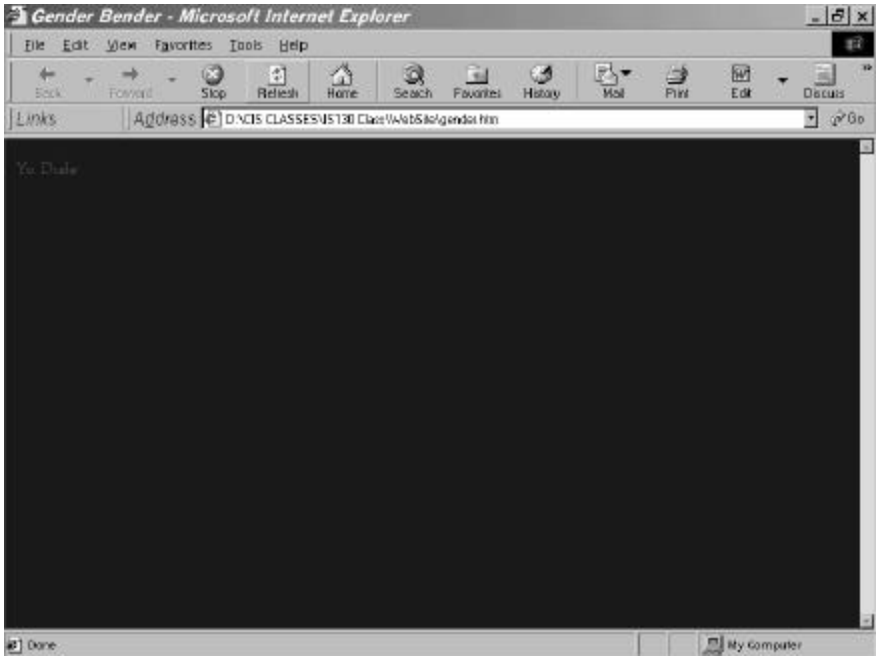


Fig ure 6-4

The Window Object

As I men tioned pre vi ously the win dow object rep re sents the browser win dow in which the script is run ning. This par tic u lar object has some really inter est-ing meth ods built in. We will explore some of them here.

One of the first things you can do with the win dow object is to open up other win dows. The openWindow method was briefly intro duced in Chap ter 4, and we will exam ine it in more detail here. In the fol low ing exam ple we open up a win dow with a pic ture dis played in it.

Example 6-2

```
<HTML>
<HEAD>
     <TITLE>Open a New Window</TITLE>
     <SCRIPT LANGUAGE="JavaScript">
     function OpenWindow()
     {
var NewWindow = window.open("advjscript.jpg", "NewWindow",
"width=350,height=400")
     }
     </SCRIPT>
</HEAD>
<BODY BGCOLOR=White>
<A HREF="javascript:OpenWindow()">Advanced Java Script</A>
</BODY>
</HTML>
```

The meat of this script is the sin gle line of code found inside the OpenWindow func tion. The first param e ter passed to this func tion is what you wish to open. This can be another HTML doc u ment or an image. The sec ond param e ter passed is the name of the vari able that is going to rep re sent this new win dow. In our case that is NewWindow. The last param e ter is the height and width of the new win dow we are open ing.

This script is fairly sim ple and easy to use. Even so, it can be very use ful. For exam ple, if you have a list of prod ucts on your web site and you do not wish to clut ter the HTML doc u ment with all the details of every prod uct, you can use this script to allow the user to view a page with those details when they click on the name of a prod uct.

The only real prob lem with our exam ple for launch ing a new win dow is that if the item you are launch ing is larger than the size param e ters you give it, there aren't any scroll bars to let you see the whole image. The next exam ple is very

sim i lar to the pre vi ous one, except that it adds some thing to the win dow.open method call to include scroll bars.

Example 6-3

```
<HTML>
<HEAD>
     <TITLE>Open a New Window</TITLE>
     <SCRIPT LANGUAGE="JavaScript">
     function OpenWindow()
     {
var NewWindow = window.open("example06-03b.htm", "NewWindow",
"width=350,height=200,scrollbars=yes")
     }
     </SCRIPT>
</HEAD>
<BODY BGCOLOR=White>
<A HREF="javascript:OpenWindow()">Advanced Java Script</A>
</BODY>
</HTML>
```

Sim ply add ing scrollbars = yes right after the dimen sion param e ters will give the new win dow you launch scroll bars! You can also add the line toolbar = yes so that your new win dow will have your browser's stan dard toolbar, if you wish.

The next exam ple com bines the scroll bars, the toolbars, and a method for the orig i nal win dow to close the win dow it launched.

Example 6-4

```
<HTML>
<HEAD>
<TITLE>Close Windows</TITLE>
<SCRIPT LANGUAGE="JavaScript">
     var childwindow = null
     function opennewwindow()
     {
          childwindow = window.open("example06-03b.htm","childwindow",
               "width=300,height=200,toolbar=yes,scrollbars=yes,")
     }
       function closenewwindow()
     {
          if (childwindow && !childwindow.closed)
               {
                    childwindow.close()
               }
     }
```

```
</SCRIPT>
</HEAD>
<BODY BGCOLOR=White>
     <CENTER>
     <H3>
          <A HREF="javascript:opennewwindow()">View Advanced JavaScript</A>
          <BR>
          <A HREF="javascript:closenewwindow()">Close Advanced JavaScript</A>
     </H3>
     </CENTER>
</BODY>
</HTML>
```

The only real dif fer ence in this script is the closenewwindow func tion. In that func tion I sim ply use the name of the vari able that rep re sents the new win dow that was launched, and call the closewindow() method. Although this dif fer-ence is small it is sig nif i cant. Allowing the user to close the child win dow from the par ent win dow that launched it is very impor tant. If you entered the code prop erly you will see some thing like this image:

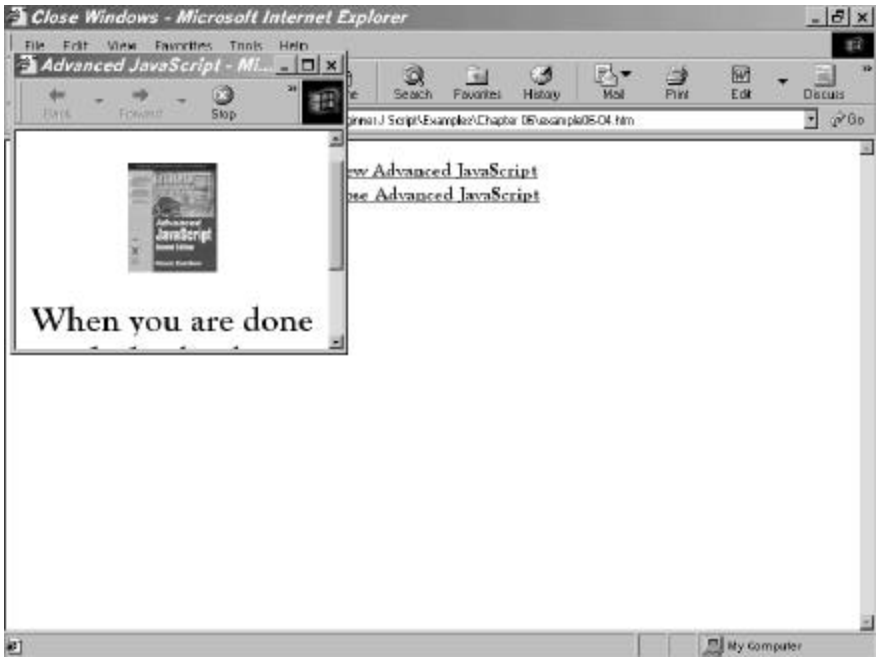


Figure 6-5

The Navigator Object

The navigator object represents the browser you are working with. From this object you can find the type of browser that is being used and the version number. Most scripts will run fine in either Microsoft Internet Explorer or Netscape Navigator. However, some scripts will not. It is a good idea, when using such scripts, to determine the browser that is being used before attempting to run a script.

The following example finds out what browser is being used and displays that information to the screen using the document object's write method.

Example 6-5

```
<HTML>
<HEAD>
     <TITLE>Browser Detection</TITLE>
</HEAD>
<BODY BGCOLOR=White>
<SCRIPT LANGUAGE="JavaScript">
var browsername= navigator.appName
var browserversion = navigator.appVersion
   if (browsername == "Microsoft Internet Explorer")
   {
         document.write("You are using MS Internet Explorer version " +
                   browserversion)
   }
   else
   {
         document.write("You are using Netscape Navigator version " +
                   browserversion)
   }
</SCRIPT>
</BODY>
</HTML>
```

This rather simple example illustrates some of the practical things you can do with the navigator object. If you entered the code properly you should see something like this:

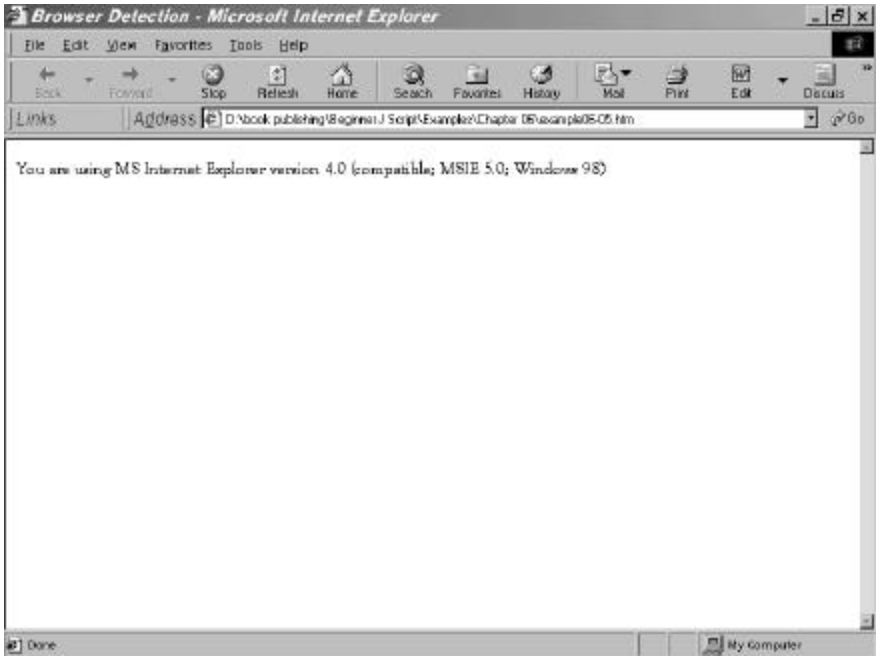


Figure 6-6

Using the History Object

At the beginning of this chapter you were introduced to the history object. Now I will show you some examples of this object in use.

Example 6-6

```
<HTML>
<HEAD>
     <TITLE>Example 06-06.htm</TITLE>
</HEAD>

<SCRIPT LANGUAGE="JavaScript">
function goback()
{
   history.back()
}

function goforward()
{
   history.forward
}
</SCRIPT>
<BODY BGCOLOR=White>
      <FORM>
```

```
        <INPUT TYPE ="button" VALUE ="Back" onClick="javascript:goback()">   |
        <INPUT TYPE ="button" VALUE ="Forward" onClick="javascript:
                        goforward()">
    </FORM>
</BODY>
</HTML>
```

This rather sim ple exam ple uses two JavaScript func tions that each con tain a sin gle line of code. Each func tion uses the his tory object to move for ward or back ward in the doc u ment's his tory. These func tions are called from HTML but tons on the body of the HTML doc u ment.

Antique Bookstore Project

With each chap ter in this book, not only will your skills grow but our antique book store pro ject grows. So now let us take what we have learned in this chap ter and add it to the antique book store. In this case we are going to add some pages describ ing some antique books. Now since this is a ficticious book store I am going to sim ply make up infor ma tion about books.

We are going to add a new page named inven tory.htm. This page will con tain the lat est addi tions to the book store's inven tory. We will have to, of course, add a link on the tool.htm page. Also each book has a very sim ple page called inventory1.htm, inventory2.htm, etc., to dis play each of the books. This page sim ply has some text and a pic ture of the book. But now for the source code for the new page.

```
<HTML>
<HEAD>
   <TITLE>Book Inventory</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT" >
     var childwindow = null
          function openbook1()
          {
               childwindow = window.open("inventory1.htm","childwindow",
                         "width=400,height=400,scrollbars=yes")
          }
          function openbook2()
          {
               childwindow = window.open("inventory2.htm","childwindow",
                         "width=400,height=400,scrollbars=yes")
          }
          function openbook3()
          {
```

```
                childwindow = window.open("inventory3.htm","childwindow",
                              "width=400,height=400,scrollbars=yes")
          }
          function openbook4()
          {
                childwindow = window.open("inventory4.htm","childwindow",
                              "width=400,height=400,scrollbars=yes")
          }
</SCRIPT
</HEAD>
<BODY   background="back1.gif" >
<P>
<CENTER>
                <H2> This months newest additions!
</CENTER>
                <P>
                <A HREF="javascript:openbook1()">View Shakespeare</A><BR>
                <A HREF="javascript:openbook2()">View Dickens</A><BR>
                <A HREF="javascript:openbook3()">View Poe</A><BR>
                <A HREF="javascript:openbook4()">View King James Bible</A><BR>
                </H2>
</BODY>
</HTML>
```

What is most interest ing about this web page is that we are using four different functions to load four different web pages. Other than that, this is the same basic launching of a new window that we saw earlier in our chapter, only applied to our antique book store!

Summary

In this chapter you have been introduced to the document, window, and navigator windows. Hopefully you are beginning to get a feel for some of the fascinating features that JavaScript offers you through these built-in objects. I would also hope that at this point you are getting comfortable with JavaScript in general.

Some of the interesting things you have learned in this chapter include the ability to open up new windows, detect what type of browser is being used, and customize the actual HTML document based on any criteria you choose.

Chapter 7

Working with Date and Time

JavaScript makes it rel a tively easy to work with date and time data via its built-in Date object. With this object you can deter mine the cur rent time, day of the week, and year. You can also change the time and date to any time or date you wish. As we move through this chap ter you will see some inter est ing appli ca tions of the meth ods of this object.

Time of Day

Rather than start ing off with some tedious details about the meth ods and prop-er ties of the Date object, let us start with an easy-to-use but inter est ing script. In this script we deter mine the time of the day and give the vis i tor to our page a greet ing based on that time of day. Let me point out that the Date method used here, getHours, returns the hours on a 24-hour clock. In other words, if it's 20 min utes after mid night, the getHours will return a zero; if it's 6 p.m., getHours will return an 18.

Example 7-1

```
<HTML>
<HEAD>
    <TITLE>Example 07-01</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var mydate = new Date()
var mytime = mydate.getHours()
if (mytime<12)
   alert("Good Morning")
else
{
     if(mytime<17)
          alert("Good Afternoon")
     else
          alert("Good Evening")
```

```
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

This script is rather sim ple but it does illus trate the essen tials of using the Date object. The first thing we do is cre ate an instance of the Date object:

```
var mydate = new Date()
```

The next thing we do is cre ate a vari able to store the cur rent time. Remem ber that this is the num ber of hours since mid night:

```
var mytime = mydate.getHours()
```

Now we have a sim ple if state ment that checks what hour it is and, based on that time of day, gives an appro pri ate greet ing. If you entered every thing correctly, you should see some thing like the image here:

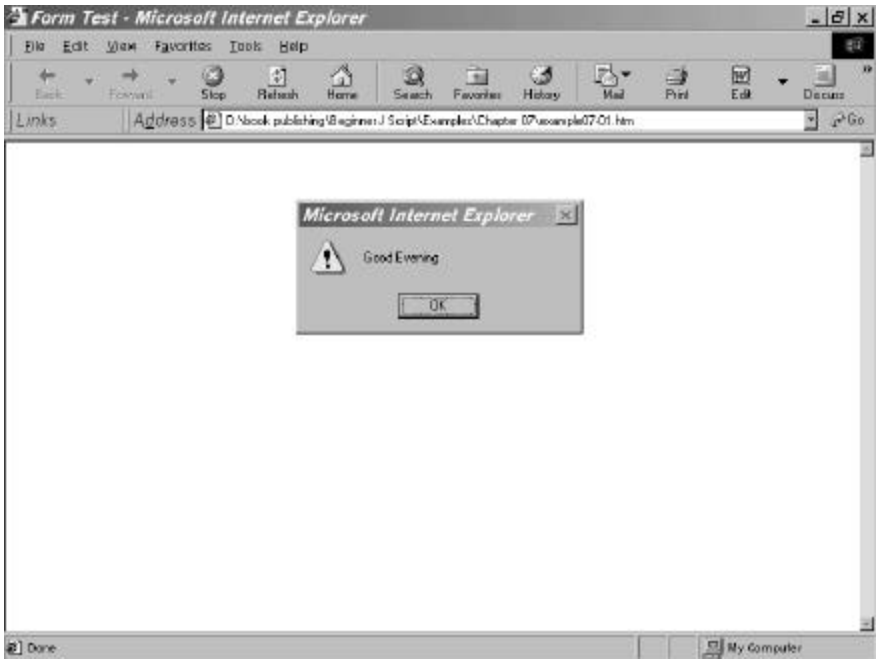


Figure 7-1

This script is fairly sim ple yet at the same time adds some inter est ing con tent to your website. A sim ple time of day greet ing can give your website a

per sonal touch and encour age more traf fic (and website traf fic is the name of the game!).

Day of Week

The next lit tle script we will exam ine is one that gives a humor ous greet ing based on the day of the week. This script uses the getDay method of the date object. This method starts at zero for Sunday and then counts for ward one by one.

Example 7-2

```
<HTML>
<HEAD>
    <TITLE>Example 07-02</TITLE>
<SCRIPT LANGUAGE="JavaScript">
 var mydate = new Date()
var myday = mydate.getDay()
if(myday==0)
    alert(“Sunday! Nap time!”)
if (myday==1)
   alert("Another Monday...Arghhh!")

if(myday==2)
   alert("Tuesday...off to a slow week!")

if(myday==3)
    alert("Wednesday...Halfway home!!!")

if (myday==4)
   alert("just 1 more day! Hang in there!")

 if (myday==5)
   alert("Thank God Its Friday!!!")
 if (myday==6)
    alert("Saturday! Break out the BBQ!")
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

This script sim ply finds out what day it is and gives a humor ous greet ing appro pri ate to the day of the week. This should illus trate to you the getDay method, just as the pre vi ous exam ple illus trates the use of the getHours

method. If you entered it cor rectly, you should see some thing like the fol lowing image:

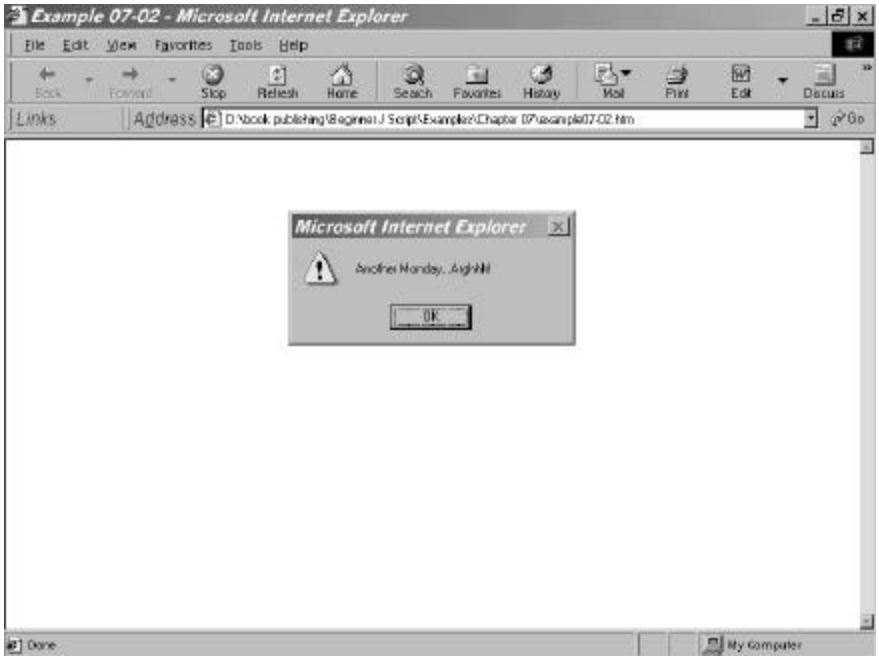


Figure 7-2

As you can prob a bly guess, there are a num ber of meth ods for get ting time and date infor ma tion from the Date object. These are getYear, getHour, getDay, getTime, getMonth, getMinutes, and getSeconds. Each of these gives a par tic-u lar out put. The fol low ing table shows the out put you can expect from each of these.

Date Attribute	Range
seconds	0-59
minutes	0-59
hours	0-23
day	0-6
date	1-31
month	0-11
year	Note: In Netscape this is the years since 1900, while in Internet Explorer it's the actual year. So 2001 will show as 101 in Netscape and as 2001 in Internet Explorer.

Now you can use these var i ous meth ods to deter mine the cur rent date, time, year, or other infor ma tion. You can also change the date and time rep re sented.

Now the first ques tion most begin ners ask is not how, but why would you want to. So let us look at an exam ple where you might want to.

Example 7-3

```
<HTML>
<HEAD>
    <TITLE>Example 07-02</TITLE>
<SCRIPT LANGUAGE="JavaScript">

var year = prompt("What year where you born")
var month = prompt("What month where you born in")
var bday =prompt("What date where you born on")
var birthDay = new Date()
birthDay.setYear(year)
birthDay.setMonth(month)
birthDay.setDate(bday)
var day = birthDay.getDay()

 if (day==0)
    alert("You were born on Sunday")

 if (day==1)
   alert("You were born on Monday")

if (day==2)
   alert("You were born on Tuesday")

if (day==3)
  alert("You were born on Wednesday")

if (day==4)
  alert("You were born on Thursday")

if (day==5)
  alert("You were born on Friday")

if (day==6)
  alert("You were born on Saturday")

if (day==0)
  alert("You were born on Sunday")
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Now take a close look at this script and let's dis cuss what it's doing. The first three lines sim ply prompt the user to enter the year, month, and day they were born. Note that these must be entered as num bers (i.e., Sep tem ber should be 9, Octo ber 10). The next four lines are where we see the code we are most inter ested in. Here we cre ate a sec ond Date object and set its year, month, and day to equal the val ues the user input. We then sim ply do a getDay() from this new Date object to find out what day of the week is produced.

Set Timeout

The setTimeout() method allows you to have some action occur after a given period of time has passed. The time is eval u ated in mil li sec onds (i.e., 5000 is 5 sec onds). This can be a very use ful func tion. Let's look at one exam ple using the setTimout() method.

Example 7-4

```
<HTML>
<HEAD>
<TITLE>Example 07-04</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function displayAlert()
{
   alert("5 seconds have elapsed since the button was clicked.")
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
Click the button on the left for a reminder in 5 seconds;
click the button on the right to cancel the reminder before
it is displayed.
<P>
<INPUT TYPE="button" VALUE="5-second reminder"
   NAME="remind_button"
   onClick="timerID = setTimeout('displayAlert()',5000)">
</FORM>
</BODY>
</HTML>
```

When you click the but ton, the event han dler's script sets a time out. The time out spec i fies that after 5,000 mil li sec onds, or 5 sec onds, the func tion

displayAlert() is called. There fore, 5 sec onds after you click the but ton an alert box is dis played.

This is not a par tic u larly excit ing script but it does show you the use of a time-out. It is a rel a tively sim ple method to use, and is fre quently used to sim ply cre ate a pause between com plex oper a tions.

Antique Bookstore Project

As with all the chap ters in this book, once we have learned some new techniques we are going to add them to our antique book store pro ject. We will sim ply add a day of week greet ing to the main.htm page. But we are going to give it a more prac ti cal twist. Rather than give a humor ous greet ing for the day of the week, we will give the store hours for that day!

Example 7-5

```
<HTML>
<HEAD>
   <TITLE>Ye Olde Book Shoppe</TITLE>
<SCRIPT LANGUAGE="JavaScript">
 var mydate = new Date()


var myday = mydate.getDay()

if(myday==0)
   alert("Sunday -Sorry we are closed today")

if (myday==1)
   alert("Monday -Sorry we are closed today")

if(myday==2)
   alert("Tuesday- Our hours today are 10 a.m. to 6 p.m.")

if(myday==3)
    alert("Wednesday- Our hours today are 10 a.m. to 6 p.m.")

if (myday==4)
   alert("Thursday- Our hours today are 10 a.m. to 6 p.m.")

 if (myday==5)
   alert("Friday - Our hours today are 10 a.m. to 7 p.m.")

 if (myday==6)
```

```
    alert("Saturday - Our hours today are 10 a.m. to 4 p.m.")

</SCRIPT>
    <SCRIPT LANGUAGE="JAVASCRIPT">
         ImageArray = new Array("banner1.gif","banner2.gif","banner3.gif")
         CurrentImage = 0
         ImageCount = ImageArray.length
        function RotateBanner()
        {
                  if (document.images)
                  {
                       CurrentImage++
                       if (CurrentImage ==ImageCount)
                       {
                            CurrentImage = 0
                       }
                       document.Banner.src=ImageArray[CurrentImage]
                       setTimeout("RotateBanner()",3000)
                  }
        }
</SCRIPT>
</HEAD>
<BODY  background="back1.gif"  onLoad="RotateBanner()">

<SCRIPT LANGUAGE="JavaScript">
    var bgOffset = 0;
    var bgObject = eval('document.body');
    function scrollbackground(maxSize)
   {
      bgOffset = bgOffset + 1;

      if (bgOffset > maxSize)
               bgOffset = 0;
       bgObject.style.backgroundPosition = "0 " + bgOffset;
  }
var scrtimer = window.setInterval("scrollbackground(50)",50);
</SCRIPT>
<CENTER>

<IMG SRC="banner1.gif"  NAME="Banner" >
<P><TABLE BORDER=1>
   <TR>
      <TD>
         <P><IMG SRC="gargshelf.gif">
     <TD>
         <P><CENTER>Ye Olde Book Shoppe</CENTER>
```

```
    <TD>
        <IMG SRC="gargshelf.gif">

</TABLE>
<P>
<P>
</CENTER>
<P>
</BODY>
</HTML>
```

In my mind the most sig nif i cant thing to notice about this web page is that we now have three JavaScripts all run ning on the same page. This is inter est ing because you will fre quently want to run more than one script on a page.

Summary

In this chap ter you have seen how to use the Date object to accom plish a variety of date and time related tasks. You should be com fort able get ting the cur rent date, year, day, or time. You should also be com fort able set ting the time and date using the setTime and setDate func tions. I believe that you will find this par tic u lar infor ma tion to be very prac ti cal for you while devel op ing websites.

Chapter 8

Working with Cookies

First we need to define what a cookie is (aside from a deli cious snack, pref er a-bly with choc o late chips!). A *cookie* is sim ply a lit tle bit of plain text data that your browser stores. This data con cerns a par tic u lar web site. Your browser stores these cook ies as unencrypted text files in a loca tion spec i fied by the browser. Once you learn to make cook ies you can have your web page store infor ma tion con cern ing a vis i tor to your site. Then when they return to your site this infor ma tion can be retrieved.

Baking Your First Cookie

Cookies are not com pli cated but the code is lengthy. The only way to do this is to just jump right in. So I am going to show you the code for a cookie and then walk you through it explain ing it line by line.

Example 8-1

```
<HTML>
<HEAD>
   <TITLE>Example 08-01</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setCookie(name, value, expires, path, domain, secure)
{
     var curCookie = name + "=" + escape(value) +
        ((expires) ? "; expires=" + expires.toGMTString() : "") +
     ((path) ? "; path=" + path : "") +((domain) ? "; domain=" + domain : "") +
     ((secure) ? "; secure" : "")
     document.cookie = curCookie
}
function getCookie(name)
{
     var prefix = name + "="
     var cookieStartIndex = document.cookie.indexOf(prefix)
           if (cookieStartIndex == -1)
```

```
          return null

     var cookieEndIndex = document.cookie.indexOf(";", cookieStartIndex+
                  prefix.length)

             if (cookieEndIndex == -1)
   cookieEndIndex = document.cookie.length

  return unescape(document.cookie.substring(cookieStartIndex +prefix.length,
              cookieEndIndex))
}
function deleteCookie(name, path, domain)
{
          if (getCookie(name))
          {
               document.cookie = name + "=" +((path) ? "; path=" + path : "") +
                   ((domain) ? "; domain=" + domain : "") +"; expires=Thu,
                   01-Jan-70 00:00:01 GMT"
          }
}
  var expiredate = new Date()
  expiredate.setTime(expiredate.getTime() + 31 * 24 * 60 * 60 * 1000)
  var name = getCookie("name")
  if (!name)
  {
       name = prompt("Please enter your name:", "John Doe")
       setCookie("name", name, expiredate)
  }

alert("Welcome back " + name)
</SCRIPT>
</HEAD>
<BODY BGCOLOR=White>
</BODY>
</HTML>
```

If you entered every thing prop erly you should see some thing like this image:

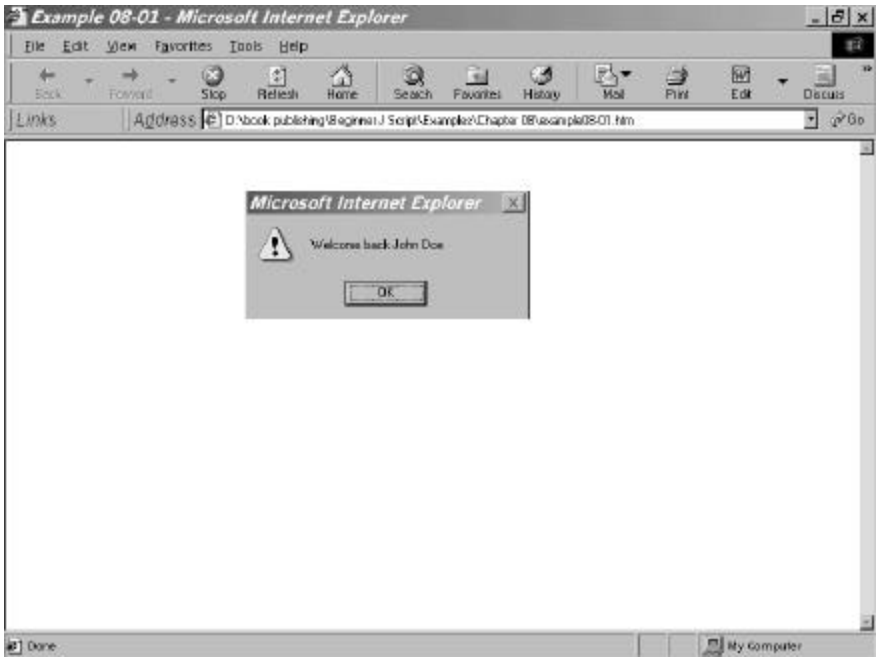


Fig ure 8-1

Now if you are begin ning to panic at the sheer size of this par tic u lar script, please don't. It only looks con vo luted. I will walk you through the source code line by line and together we will sort this out. Let us begin by look ing at the first func tion you see, SetCookie.

SetCookie

The syn tax is:

```
function setCookie(name, value, expires, path, domain, secure)
```

First note that this func tion takes six param e ters. Your code will only sup ply the first three. The last three are deter mined by the browser. The first param-e ter is called name. It is sim ply the name of the cookie.

It's a good idea to pick a name that is rel e vant to the type of data the cookie holds. The next param e ter is the value or con tent that cookie will hold. We then have expires, which holds an expi ra tion date for the cookie. If you do not set an expi ra tion date, the cookie will expire the next time you reboot your PC. This is referred to as a *pre-expired cookie.*

Even though you don't sup ply the final three param e ters it's a good idea to under stand what they are. The fourth param e ter sim ply is the path to where your browser stores cook ies. The fifth param e ter indi cates what domain the

cook ies are asso ci ated with. A domain can have a max i mum of 20 cook ies. Finally we have the secure param e ter, which sim ply tells us if this is a secure cookie or not.

Next the code sim ply cre ates a vari able to con tain the cookie infor ma tion and con cat e nates it all together. Finally we use the doc u ment object's cookie property to add this cookie to the user's cookie list.

```
var curCookie = name + "=" + escape(value) +
    ((expires) ? "; expires=" + expires.toGMTString() : "") +
    ((path) ? "; path=" + path : "") +((domain) ? "; domain=" + domain : "") +
    ((secure) ? "; secure" : "")
    // Actually placing the cookie
    document.cookie = curCookie
```

What this func tion accom plishes is that it takes the val ues you pass it and the val ues that the browser itself passes it, and it cre ates a cookie to store that data.

GetCookie

The next func tion we exam ine is the code for the GetCookie func tion. This is called to see if a valid cookie exists for this web page vis i tor. If it does, the func tion will sim ply return a null value. At that point you can prompt the user to enter the data, and then call setCookie(). How ever, if it does find a match, you can then use that data.

The func tion begins with this line of code:

```
var prefix = name + "="var cookieStartIndex = document.cookie.indexOf(prefix)
```

This code seg ment is cre at ing a vari able named pre fix. That vari able is set to equal the start ing point, in the cookie file, of the cookie name that was passed to this func tion. This func tion will use the doc u ment object's cookie pro ject to see if it can find this cookie. If it can, then it will return the index, and the cookie's value can be read in. If not, it will return a null. In fact, the very next line takes care of this:

```
if (cookieStartIndex == -1)
     return null
```

If cookieStartIndex can not be found, sim ply have the func tion return a null, indi cat ing that no cookie was found.

The next few lines try to find the end ing point of this cookie. If one can not be found, it is assumed that this is the last cookie your browser has in its cookie list and the end of this list is assumed to be the end of this cookie:

```
var cookieEndIndex = document.cookie.indexOf(";", cookieStartIndex+
                prefix.length)

    if (cookieEndIndex == -1)
         cookieEndIndex = document.cookie.length
```


Now once the func tion has deter mined that the cookie exists and that it has a start point as well as an end ing point, then it's time to sim ply return the contents of that cookie. This is what the final line of the func tion accom plishes:

```
return unescape(document.cookie.substring(cookieStartIndex +prefix.length,
cookieEndIndex))
```

The third func tion, deleteCookie(), sim ply deletes a cookie if you pass it the cookie name. In the pre vi ous exam ple this func tion was not actu ally called, but I thought I should show it to you in case you need it later.

Calling the Functions

Now how do we actu ally use these func tions to get cook ies and to cre ate them? Well, the last few lines of the script accom plish this, so let us give them a look. First we have two lines of code that cre ate a Date object and incre ment its value:

```
var expiredate = new Date()
expiredate.setTime(now.getTime() + 31 * 24 * 60 * 60 * 1000)
```

This expiredate object will be used to set the expi ra tion date of our cookie. You can use the date func tions to set this expi ra tion date to any date you wish. I just picked this par tic u lar date for illus tra tion pur poses.

Now that we have an expi ra tion date (and that will only be used when we set a new cookie, not when we get a cookie) we will now see if the cookie we want already exists on the web page vis i tor's machine.

```
var name = getCookie("name")
```

This code is really quite sim ple, but effec tive. We sim ply call the getCookie func tion and find out what it returns for the name cookie. If it finds a valid name, then we can sim ply use that name. If not, then we will get a null for a return value, in which case we can sim ply prompt the user to enter their name then use the setCookie func tion to cre ate a cookie hold ing their name:

```
if (!name)
{
     name = prompt("Please enter your name:", "John Doe")
     setCookie("name", name, expiredate)
}
alert("Welcome back " + name)
```

And that, in a nut shell, is how our cookie script works. It is really not that com pli cated, it is sim ply long. So in order to make sure you do under stand the con cepts, let's exam ine another script that stores a some what dif fer ent cookie.

Bake Another Cookie

This cookie works sim i lar to the pre vi ous one, and I hope that by exam in ing both cook ies you will get a good under stand ing of how cook ies work. This script asks the user what back ground color they would pre fer when view ing the web page, and saves that value in a cookie. Then any time they visit the web page, it uses the back ground color they selected.

Example 8-2

```
<HTML>
<HEAD>
<TITLE>Example08-02</TITLE>
<HEAD>
<SCRIPT LANGUAGE = "JavaScript">
var expiredate= new Date();
expiredate.setTime(expiredate.getTime() + (30*24*60*60*1000))
  var backcolor = getCookie("bgcolor")
  if (backcolor == null)
  {
    backcolor = prompt("What is your favorite background color?")
    setCookie("bgcolor", backcolor, expiredate)
  }
  document.bgColor=backcolor
function setCookie(name, value, expires, path, domain, secure)
{
     var curCookie = name + "=" + escape(value) +
        ((expires) ? "; expires=" + expires.toGMTString() : "") +
        ((path) ? "; path=" + path : "") +
        ((domain) ? "; domain=" + domain : "") +
        ((secure) ? "; secure" : "")
        document.cookie = curCookie
}
function getCookie(cookiename)
```

```
{
 var prefix = cookiename + "="
 var cookieStartIndex = document.cookie.indexOf(prefix)
          if (cookieStartIndex == -1)
               return null
          var cookieEndIndex = document.cookie.indexOf(";", cookieStartIndex+
                    prefix.length)
          if (cookieEndIndex == -1)
               cookieEndIndex = document.cookie.length

          return unescape(document.cookie.substring(cookieStartIndex
                    +prefix.length, cookieEndIndex))
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```


If you entered the code prop erly then you should be able to see the fol low ing image in your browser:

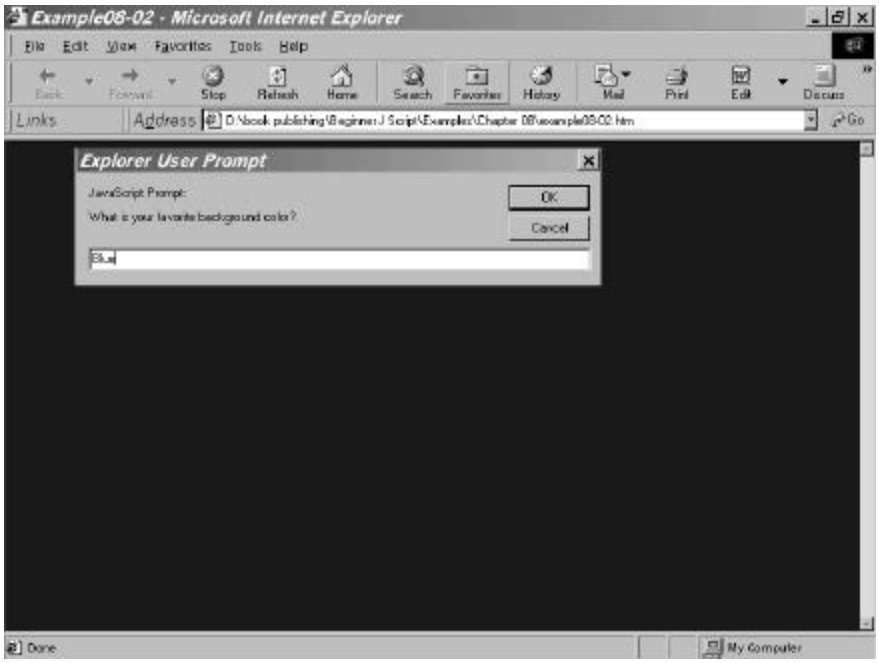


Fig ure 8-2

Hope fully you notice sev eral sim i lar i ties between this script and the last one. But let us go over this one together, just to make sure you under stand everything that is hap pen ing. The first dif fer ence is merely struc tural. In this script I placed the call to the func tions first, then the func tion def i ni tions. This is the oppo site of the order used in the pre vi ous script. I did this on pur pose to show you that the order in which you do these actions really makes no dif fer ence.

Let's take a look at the first few lines of code and see what they are doing:

```
var expiredate= new Date();
expiredate.setTime(expiredate.getTime() + (30*24*60*60*1000))
var backcolor = getCookie("bgcolor")
```

First we cre ate a date object called expiredate. As you might guess, this is going to hold the expi ra tion date for the cookie. We then set the expi ra tion date for some time beyond the cur rent time. Next we call the getCookie function to retrieve the con tents of the cookie named bgcolor. What ever is returned is placed in the vari able backcolor. If the cookie is found, then backcolor should con tain some color. If no match ing cookie is found, then we will get a null value in the vari able backcolor.

Next we check to see if the value of backcolor is null. If so, we are going to prompt the user to enter their pre ferred color and then cre ate a cookie to save that infor ma tion. We will then set the back ground color of the cur rent web page to the color the user has selected:

```
if (backcolor == null)
{
     backcolor = prompt("What is your favorite background color?")

     setCookie("bgcolor", backcolor, expiredate)
}
document.bgColor=backcolor
```

We do not need to closely exam ine the 'setCookie' and 'getCookie' func tions because they are exactly iden ti cal to the ones used in the pre vi ous func tion. This illus trates a very impor tant point. The pro cess of cre at ing a cookie, or retriev ing one, is the same regard less of what you choose to name a cookie or what value you place in the cookie. This means you can use these same cookie scripts, with some minor alter ations, to cre ate cook ies to save any data you might wish to save.

Technical Note: The way a client's browser communicates with the web server is via the HTTP (HyperText Transfer Protocol) protocol. When a user requests a specific page, the browser sends that request to the web server. This activity is all transparent to the web page user. Among the items requested, an HTTP request includes a header that defines the most important attributes, such as the URL of the requested page. An HTTP request also includes the cookies.

The server then returns an HTTP response. This response also has a header that contains valuable information. The general structure of an HTTP header is as follows:

Field-name: *Information*

When the server returns an HTTP object to the client, it may also transmit some state information for the client, to store as cookies. Since a cookie is basically simple text, the server-side script does not have the ability to abuse the client machine in any way. In addition to its textual value, a cookie contains several attributes, such as the range of URLs for which the cookie is valid. Any future HTTP requests from the client to one of the URLs in the above range will transmit back to the server the current cookie's value on the client.

An HTTP cookie is introduced to the client in an HTTP request using the following syntax:

```
Set-Cookie: NAME=VALUE; expires=DATE; path=pathName; domain=DOMAIN_NAME; secure
```

The attributes are as follows:

name=value

This simply specifies the name of the cookie and the value to be stored in the cookie. Technically, this is the only field that is required. All others are purely optional.

expires=date

As you probably guessed, this attribute simply sets an expiration date for the cookie. This field is optional. However, if no expiration date is set, a cookie will expire the next time the client machine is rebooted. The date string is formatted as follows:

Wdy, DD-Mon-YYYY HH:MM:SS
domain

The domain attribute makes sure that only hosts within the specified domain can set a cookie for the domain. Domains must have at least two or three periods, to avoid collision between domains of the form .com,

.edu, etc. There are seven com mon top-level domains that require at least two peri ods in their domain name: com, edu, net, org, gov, mil, and int. All other domains require at least three peri ods in their domainName. An exam ple of a domain name would be www.wordware.com.

The default value of domain is the host name of the server which gen er-ated the cookie response.

path=pathName

Path spec i fies a sub set of URLs in a domain for which a cookie is valid. Basically once you have found an appro pri ate domain, you have to find the indi vid ual website within that domain that is rel e vant to this cookie.

secure

If a cookie is marked secure, it will only be trans mit ted across a secured com mu ni ca tion chan nel between the cli ent and the host.

Antique Bookstore Project

As you might guess, we are going to add this script to our antique book store pro ject. We are going to use the first cookie script on the main.htm page of our pro ject. This way we can retain the name of cus tom ers who visit our site and greet them when they return. As in each chap ter when we add some thing to a page in our pro ject I show you the com plete code, includ ing the pre vi ous code. This allows you to see the script in the con text of the other HTML and other scripts already on the web page.

```
<HTML>
<HEAD>
   <TITLE>Ye Olde Book Shoppe</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setCookie(name, value, expires, path, domain, secure)
{
   var curCookie = name + "=" + escape(value) +
        ((expires) ? "; expires=" + expires.toGMTString() : "") +
   ((path) ? "; path=" + path : "") +
   ((domain) ? "; domain=" + domain : "") +
   ((secure) ? "; secure" : "")
   document.cookie = curCookie
}
function getCookie(name)
{
var prefix = name + "="
   var cookieStartIndex = document.cookie.indexOf(prefix)
```

```
          if (cookieStartIndex == -1)
               return null

          var cookieEndIndex = document.cookie.indexOf(";", cookieStartIndex+
                         prefix.length)

          if (cookieEndIndex == -1)
               cookieEndIndex = document.cookie.length

          return unescape(document.cookie.substring(cookieStartIndex
                         +prefix.length, cookieEndIndex))
}

function deleteCookie(name, path, domain)
{
          if (getCookie(name))
          {
          document.cookie = name + "=" +((path) ? "; path=" + path : "") +
                  ((domain) ? "; domain=" + domain : "") +"; expires=Thu,
                   01-Jan-70 00:00:01 GMT"
          }
}
  var expiredate= new Date()
  expiredate.setTime(expiredate.getTime() + 30 * 24 * 60 * 60 * 1000)
  var name = getCookie("name")
  if (!name)
  {
       name = prompt("Please enter your name:", "John Doe")
      setCookie("name", name, expiredate)
 }

alert("Welcome back " + name)
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
var mydate = new Date()


var myday = mydate.getDay()

if(myday==0)
   alert("Sunday -Sorry we are closed today")

if (myday==1)
   alert("Monday -Sorry we are closed today")

if(myday==2)
```

```
   alert("Tuesday- Our hours today are 10 a.m. to 6 p.m.")

if(myday==3)
    alert("Wednesday- Our hours today are 10 a.m. to 6 p.m.")

if (myday==4)
   alert("Thursday- Our hours today are 10 a.m. to 6 p.m.")

 if (myday==5)
   alert("Friday - Our hours today are 10 a.m. to 7 p.m.")

 if (myday==6)
    alert("Saturday - Our hours today are 10 a.m. to 4 p.m.")

</SCRIPT>
    <SCRIPT LANGUAGE="JAVASCRIPT">
         ImageArray = new Array("banner1.gif","banner2.gif","banner3.gif")
         CurrentImage = 0
         ImageCount = ImageArray.length
        function RotateBanner()
        {
                 if (document.images)
                 {
                       CurrentImage++
                       if (CurrentImage ==ImageCount)
                       {
                            CurrentImage = 0
                       }
                            document.Banner.src=ImageArray[CurrentImage]
                            setTimeout("RotateBanner()",3000)
                       }
                 }


</SCRIPT>
</HEAD>
<BODY  background="back1.gif"  onLoad="RotateBanner()">

<SCRIPT LANGUAGE="JavaScript">
    var bgOffset = 0;
    var bgObject = eval('document.body');
    function scrollbackground(maxSize)
   {
      bgOffset = bgOffset + 1;
      if (bgOffset > maxSize)
               bgOffset = 0;
```

```
        bgObject.style.backgroundPosition = "0 " + bgOffset;
  }
var scrtimer = window.setInterval("scrollbackground(50)",50);
</SCRIPT>
<CENTER>
<IMG SRC="banner1.gif"  NAME="Banner" >
<P><TABLE BORDER=1>
   <TR>
      <TD>
         <P><IMG SRC="gargshelf.gif">
     <TD>
         <P><CENTER>Ye Olde Book Shoppe</CENTER>
     <TD>
         <IMG SRC="gargshelf.gif">
</TABLE>
<P>
<P>
</CENTER>
<P>
</BODY>
</HTML>
```


I hope you notice that we are now run ning five sep a rate scripts on the same web page. Usually this would be dis cour aged, as it can make the web page appear too “busy.” How ever, in this case it was done pur pose fully to illus trate a point. You can eas ily have mul ti ple scripts on a sin gle page. You sim ply need to place each script inside of its own <SCRIPT> </SCRIPT> tags.

Summary

In this chap ter you were shown a very use ful appli ca tion of JavaScript: cookies. Cookies are the most com mon way of retain ing data about a website vis i tor so that the data can be retrieved the next time the per son vis its that website. This not only cre ates a more user-friendly web page, but it can save the user the trou ble of hav ing to re-enter key data every sin gle time he or she vis its the web site.

Chapter 9

Working with the Status Bar

The sta tus bar is found at the bot tom of the browser's win dow. Both Nav i ga tor and Internet Explorer have a sta tus bar. It usu ally dis plays the cur rent sta tus of the doc u ment being loaded.

It can be use ful for you to occa sion ally dis play infor ma tion in the sta tus bar. For tu nately this is very easy in JavaScript. In this chap ter you will see a variety of ways you can dis play infor ma tion in the sta tus bar. Sim ply access ing the sta tus prop erty of the win dow object gives you access to the sta tus bar.

Image Data

One very easy tech nique is to dis play infor ma tion in the sta tus bar when a person passes the mouse over some image.

Example 9-1

```
<HTML>
<HEAD>
   <TITLE>Example 09-01</TITLE>
</HEAD>
<BODY BGCOLOR=white>
<CENTER>
<A HREF="somepage.htm"
    onMouseOver= " window.status='This is a link to some page'
    return true" onMouseOut="window.status= ' '
    return true">First Link</A> |

 <A HREF="someotherpage.htm"
    onMouseOver="window.status='This is a link to some other page'
    return true"  onMouseOut="window.status='Please dont go '
    return true">Second Link </a>
```

```
</CENTER>
</BODY>
</HTML>
```

If you enter the code exactly as it appears above you should see the following images in your browser. When the page first loads you will see this:

Figure 9-1

Then if you move your mouse over the first link, you will see a message in the status bar.

Figure 9-2

If you move your mouse over the second link, you will see a message in the status bar. With this link (unlike the first one) you will get another message when you move off of the link.

Figure 9-3

By closely exam in ing the sec ond link we can see clearly how this works. It begins with a stan dard HTML ref er ence < A HREF = “somepage.htm”. But before we close the tag we have some other code.

The next line we see is onMouseOver = win dow.sta tus = 'This is a link to some page'. What this is say ing is sim ply that when a mouse moves over this link to place the text you see in the sta tus bar of the browser. An impor tant note to remem ber is that in your text you can not have any apos tro phes. Since the text itself is enclosed in apos tro phes it will cause an error.

After this we have two choices for the onMouseOut event. We can sim ply write win dow.sta tus = …. as we did with the first link, or we can place some other text in the sta tus bar when the mouse moves off of the link.

This par tic u lar lit tle trick is not hard to use at all, and it adds an impor tant dimen sion to your website. Each of your links can now dis play more details in the sta tus bar. This can be very use ful for vis i tors to your site.

T-Banners

The T-banner sim u lates a type writer. It dis plays each mes sage by typ ing it in, one char ac ter at a time. It seems as if some one is typ ing the mes sage at a certain speed, delet ing it upon com ple tion. First, take a look at the script:

Example 9-2

```
<HTML>
<HEAD>
<TITLE>Example 09-02</TITLE>
<SCRIPT LANGUAGE="JavaScript">

var speed = 100  // this speed value actually works in reverse; lower the
                 // number to increase speed

var pause = 1000 // increase value to increase pause
// set initial values
var timerID = null
var bannerRunning = false
var offset = 0
var text = "How do you like this T-Banner"

function stopBanner()
{
```

```
      // if banner is currently running
      if (bannerRunning)
       // stop the banner
       clearTimeout(timerID)
       bannerRunning = false
}

function startBanner()
{
     stopBanner()
     showBanner()
}

function showBanner()
     {


          if (offset < text.length)
               {
                    var partialMessage = text.substring(0, offset + 1)
                    // display partial message in status bar
                    window.status = partialMessage
                    // increment index of last character to be displayed
                    offset++

                    // recursive call after specified time
                    timerID = setTimeout("showBanner()", speed)
                    // banner is running
                    bannerRunning = true
               }
               else
               {
                    offset = 0
                    timerID = setTimeout("showBanner()", pause)
                    bannerRunning = true
               }
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="startBanner()">
</BODY>
</HTML>
```

When you type in this script and run it in your browser, you should see something like the following:

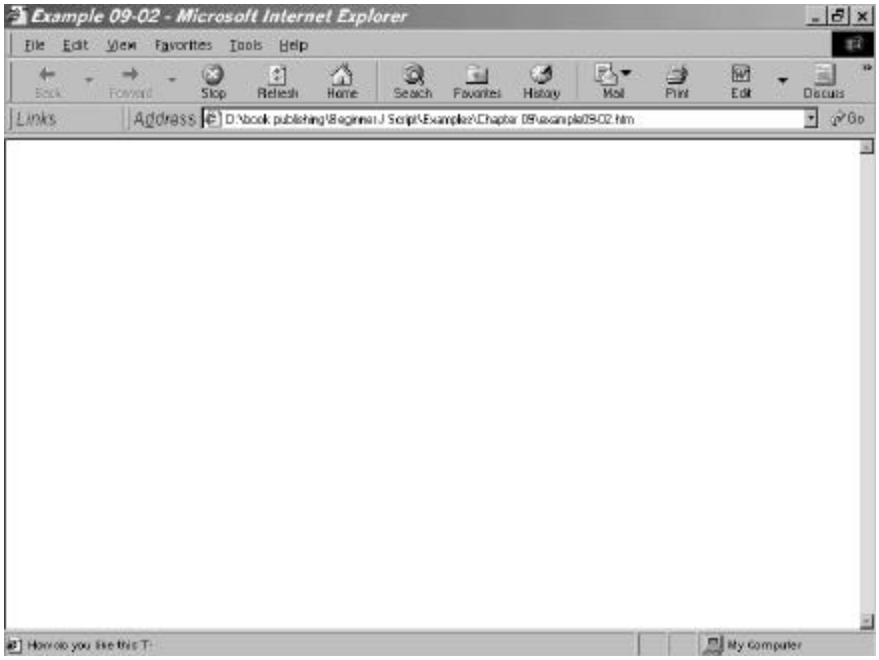


Figure 9-4

This script is actually rather simple. In fact, it is its simplicity that makes it work. The first portion of the script merely declares the variables that we will need throughout the rest of the script.

```
var speed = 100  // this speed value actually works in reverse; lower the
                 // number to increase speed
var pause = 1000 // increase value to increase pause
// set initial values
var timerID = null
var bannerRunning = false
var offset = 0
var text = "How do you like this T-Banner"
```

The first variable simply tells us how fast to put the banner in the status bar. The second tells us how much pause before running the banner again. The third variable is a timer variable that we initialize to a null value. The bannerRunning variable lets us track whether the banner is currently running or not. The offset variable is really key to the whole operation, as you will see shortly. The final variable is the text variable that simply contains whatever text we wish to display.

The stopBanner func tion merely causes the ban ner to stop dis play ing. The startBanner func tion ensures that the ban ner is not cur rently run ning (by call-ing the afore men tioned stopBanner func tion), then calls the showBanner func tion. It is this last func tion with which we are con cerned. It is here that the real action takes place in this script.

```
if (offset < text.length)
{
     var partialMessage = text.substring(0, offset + 1)
     // display partial message in status bar
     window.status = partialMessage
           offset++
     // recursive call after specified time
     timerID = setTimeout("showBanner()", speed)
     // banner is running
     bannerRunning = true
}
```


Let us exam ine this seg ment of script and make sure that you under stand what it is doing. The off set vari able, which you were intro duced to pre vi ously, tells us which char ac ter in the text vari able we are cur rently at. So the first thing we do is see if that value is still less than the length of the total text value. If it is then we exe cute the remain ing code. The secret of the script is in just three lines:

```
var partialMessage = text.substring(0, offset + 1)
window.status = partialMessage
      offset++
```

What hap pens here is we grab the seg ment of the text vari able from zero to the value of the off set + 1 and then we dis play that text in the sta tus bar. On the first pass we will sim ply dis play the first char ac ter, then the sec ond char ac-ter, then the third, and so on. Although it appears that the char ac ters are being typed across the sta tus bar, what is really hap pen ing is that the sta tus bar's value is com pletely chang ing each time to reflect a lon ger seg ment of the text!

After this, the func tion just keeps call ing itself until the entire text is dis-played. You should note that the pro cess of hav ing a func tion call itself is referred to as *recursion.* A func tion that does call itself is said to be a *recursive function.*

Antique Bookstore Project

As with all the chap ters in this book, we now need to add some of what we have learned to our ongo ing antique book store pro ject. We are actu ally going to add two things. We will begin by using the first exam ple I showed you to add some infor ma tional con tent to our inven tory page. The fol low ing shows only the code we are add ing here.

Inventory.htm

```
<HTML>
<HEAD>
   <TITLE>Book Inventory</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT" >
var childwindow = null
function openbook1()
{
childwindow = window.open("inventory1.htm","childwindow",
"width=400,height=400,scrollbars=yes")
}
function openbook2()
{
childwindow = window.open("inventory2.htm","childwindow",
"width=400,height=400,scrollbars=yes")
}
function openbook3()
{
childwindow = window.open("inventory3.htm","childwindow",
"width=400,height=400,scrollbars=yes")
}
function openbook4()
{
childwindow = window.open("inventory4.htm","childwindow",
"width=400,height=400,scrollbars=yes")
}
</SCRIPT
</HEAD>
<BODY   background="back1.gif" >
<P>
<CENTER>
               <H2> This month's newest additions!
</CENTER>
<P>
<A HREF="javascript:openbook1()"
onMouseOver= " window.status='We have three copies of this book All in fine
condition'
```

```
return true" onMouseOut="window.status= ' '
return true">View Shakespeare</A><BR>

<A HREF="javascript:openbook2()"
onMouseOver= " window.status='We just aquired this book last month'
return true" onMouseOut="window.status= ' '
return true">View Dickens</A><BR>

<A HREF="javascript:openbook3()"
onMouseOver= " window.status='This book is in fair conditon'
return true" onMouseOut="window.status= ' '
return true">View Poe</A><BR>

<A HREF="javascript:openbook4()"
onMouseOver= " window.status='We have 1 copy in mint condition'
return true" onMouseOut="window.status= ' '
return true">View King James Bible</A><BR>
</H2>
</BODY>
</HTML>
```


Summary

In this chap ter you have seen that it is really quite easy to manip u late the status bar to dis play infor ma tion. I have shown you a few direct exam ples to illus trate this point. You will find that the min i mal effort required in implement ing these tech niques is well worth it. Your web site vis i tors will be able to get addi tional infor ma tion in the sta tus bar with out you hav ing to add unnec-es sary clut ter to your web page.

Chapter 10

Creating Dynamic Menus

The vis i tors to your website are prob a bly used to desk top appli ca tions (running on their PC or Mac) that allow them to nav i gate the soft ware via a wide range of menus. I am sure that you have used drop-down menus, pop-up menus, and other type of menus as well. For tu nately it is not par tic u larly dif ficult to place such menus in your web page using JavaScript.

In order to cre ate many of these menus we have to depend on a web tech nology called Cas cading Style Sheets. Since this book is about JavaScript we don't go into depth on how exactly Cas cading Style Sheets work; we just use them in con junc tion with our stan dard HTML and JavaScript. It is entirely pos si ble to use these scripts with only a very cur sory knowl edge of Style Sheets.

Pop-Up Menus

The first menu type we will exam ine pro duces a sim ple pop-up menu when the user moves the mouse over a par tic u lar seg ment of text. Let us look at a sam ple and exam ine it to under stand how it works.

Example 10-1

```
<HTML>
<HEAD>
   <TITLE>ex10-01</TITLE>
   <STYLE TYPE="text/css">
       . popup {border-width:4; border-style: solid; border-color:blue;
             position:absolute; background-color:yellow; visibility:hidden}
   </STYLE>
   <SCRIPT>
     function setUp()
     {
          makepopup ("Item1" ,280, "This is the popup for item 1")
          makepopup ("Item2",240, "Hey this is another popup.")
```

```
            makepopup ("Item3",280, "Last, but not least, popup3.")
        }
        function makepopup (id, width, message)
        {
            var htmltext = '<STYLE
               TYPE="text/css">#'+id+'{width:'+width+';}</STYLE>';
               htmltext +='<DIV CLASS="popup" id="'+id+'">'+message+'</DIV>';
               document.write(htmltext);
        }

        function show(id, event)
        {
            document.all[id].style.pixelLeft = (document.body.scrollLeft
                    +event.clientX) + 10;
            document.all[id].style.pixelTop = (document.body.scrollTop +
                    event.clientY) + 10;
            document.all[id].style.visibility="visible";
        }


        function hide(id)
            {
                 document.all[id].style.visibility="hidden";
            }


</SCRIPT>
</HEAD>
<BODY BGCOLOR=white>
<CENTER>
    List of Items
    <UL>
       <LI>
           <a href="" onMouseOver="show('Item1', event)"
                       onMouseOut="hide('Item1')">Item 1</a>
       <LI>
           <a href="" onMouseOver="show('Item2', event)"
                       onMouseOut="hide('Item2')">Item 2</a>
       <LI>
           <a href="" onMouseOver="show('Item3', event)"
                       onMouseOut="hide('Item3')">Item 3</a>
    </UL>

<script language="JavaScript">
    setUp()
</script>
```

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

This script may look a bit com pli cated but don't worry, it's not. The first new item you may notice is the <STYLE> tag. This is used with Cas cading Style Sheets (CSS) and we use it to cre ate the lit tle bal loon. It is com mon to use these Cas cading Style Sheets in con junc tion with a script ing lan guage such as JavaScript. Essen tially its param e ters define the way in which the pop-up win-dow will look. After that, the rest of the script is fairly stan dard JavaScript. Let's take a closer look

The first func tion you see, setUp, sim ply defines the param e ters the pop-up menu will have. This includes what text it will dis play and its width. This func-tion in turn uses makepopup to cre ate the pop-up win dow with the param e ters you put in setUp.

The func tion Show() actu ally dis plays the pop-up win dow. This func tion is called when the mouse moves over the text in the body of your HTML. The hide func tion sim ply hides the pop-up win dow and is called when the mouse moves off of the text in the body of your HTML. Both of these func tions uti lize ele ments of the Cas cading Style Sheet, par tic u larly the vis i ble prop erty.

If you entered the code cor rectly, you should be able to use your browser to view some thing sim i lar to the fol low ing:

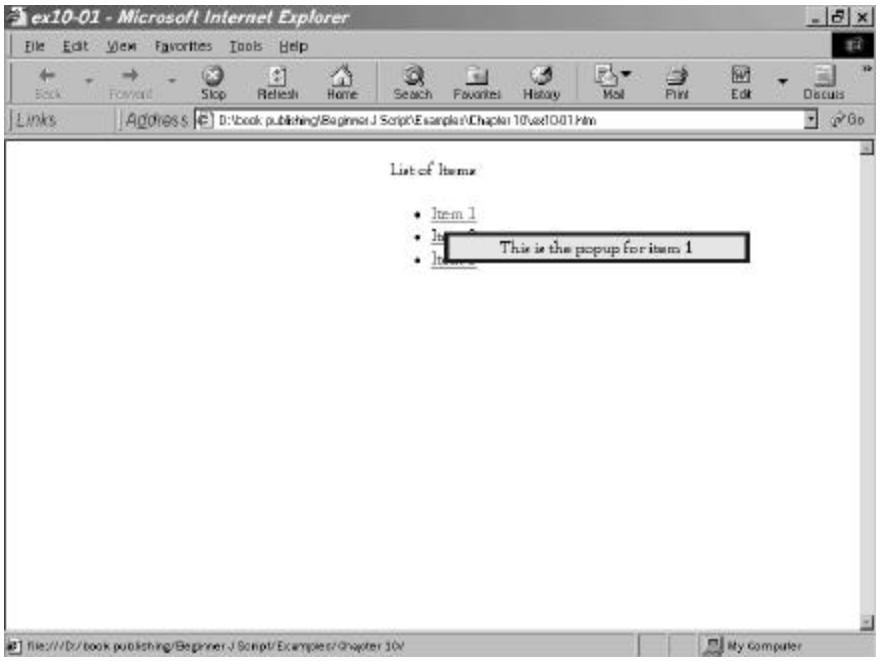


Fig ure 10-1

Drop-Down Menus

The drop-down menu sim ply uses an HTML choice form ele ment. This looks like a drop-down or combo box in Microsoft Win dows. You use this form element to pro vide the user with a list of choices, then take their choice and pass it to your JavaScript. Let's look at an exam ple.

Example 10-2

```
<HTML>
<HEAD>
<TITLE> Example 10-02</TITLE>
<SCRIPT LANGUAGE = "JavaScript">

function movetourl()
{
window.location=document.dropmenu.website.options[document.dropmenu.website.sele
ctedIndex].value
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="dropmenu">
            <SELECT NAME="website">
            <OPTION VALUE=""><B>Choose your link</b>
            <OPTION VALUE="http://www.wordware.com">WordWare
            <OPTION VALUE="http://www.amazon.com">Amazon Books
            <OPTION VALUE="http://www.nortexsolutions.com">Nortex Solutions
            </SELECT>
            <INPUT TYPE="button" VALUE="Go" onClick="movetourl()">
</FORM>
</BODY>
</HTML>
```

If you enter all the code cor rectly, you should be able to see some thing like the following illustration:

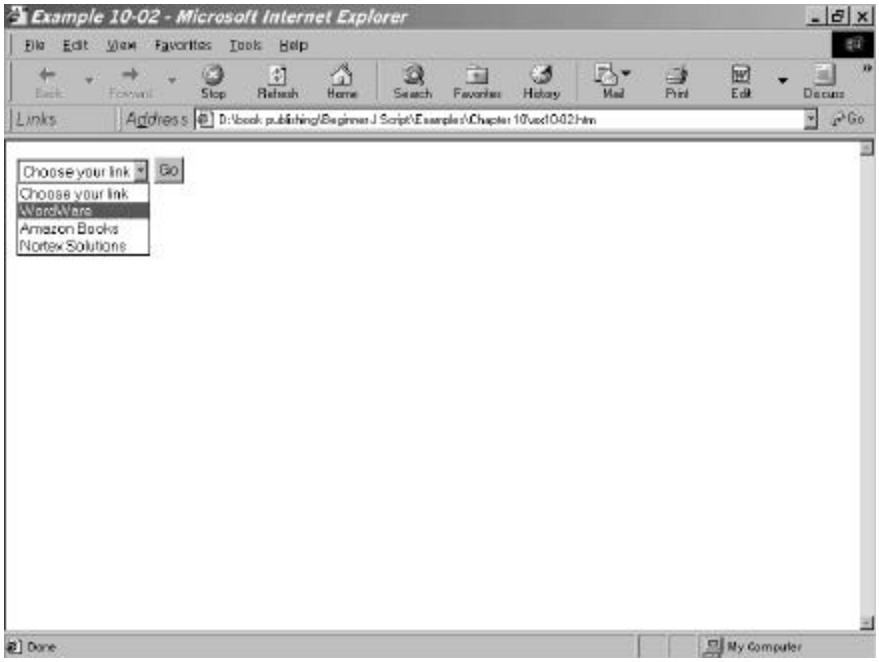


Fig ure 10-2

Let's take a look at the code and see what is hap pen ing. The actual script itself is amaz ingly sim ple. It is a sin gle line of code:

```
window.location=document.dropmenu.website.options[docu-
ment.dropmenu.website.selectedIndex].value
```

It uses the win dow object's loca tion prop erty to sim ply move the web page to the URL spec i fied in the menu. This is pretty straight for ward and easy to follow. The way we actu ally cre ate the menu is using HTML form ele ments. In this case we use the select ele ment that cre ates a drop-down menu. When the user has selected the web page they wish to go, to they then click on the Go but ton and from its onClick event we call our JavaScript. This par tic u lar menu is very easy to imple ment, as you can see.

Pull-Down Menus

The next type of menu we will dis cuss in a pull-down menu. This menu works like many appli ca tions in Microsoft Win dows. With this menu exam ple only the top-level menu title appears on the screen. When your mouse moves over that menu title the rest of the menu drops down into view. This is a very interest ing effect. Let's look at an exam ple.

Example 10-3

```
<HTML>
<HEAD>
<TITLE>Example 10-03</TITLE>
<SCRIPT LANGUAGE="JavaScript">
    var browser
    if (document.getElementById)
     {
            browser = true
     }
     else
    {
           browser =  false
    }
    function toggleMenu(currElem,nextPos)
         {
              menuObj = (browser ) ? document.getElementById(currElem).style :
                        eval("document." + currElem)
              if (toggleMenu.arguments.length == 1)
              {
                   nextPos = (parseInt(menuObj.top) == -5) ? -90 : -5
              }
                   menuObj.top = (browser ) ? nextPos + "px" : nextPos
        }

</SCRIPT>

<STYLE TYPE="TEXT/CSS">
    .menu {position:absolute; font:12px arial, helvetica, sans-serif;
             background-color:tan; layer-background-color:tan; top:-90px}
    #fileMenu {left:10px; width:70px}
    #searchMenu {left:85px; width:100px}
    A {text-decoration:none; color:blue}
    A:hover {background-color:blue; color:red}

</STYLE>
</HEAD>
<BODY BGCOLOR="white">
<DIV ID="fileMenu" CLASS="menu" onMouseover="toggleMenu('fileMenu',-5)"
onMouseout="toggleMenu('fileMenu',-90)"><BR>
     <A HREF="javascript:window.open()">Open</A><BR>
     <A HREF="javascript:history.back()">Back</A><BR>
     <A HREF="javascript:history.forward()">Forward</A><BR>
     <A HREF="javascript:window.close()">Close</A><HR>
     <A HREF="javascript:changemenu('fileMenu')">File</A>
</DIV>
```

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

If you enter every thing prop erly and then open up this web page in your browser, you should see some thing like the fol low ing fig ures.

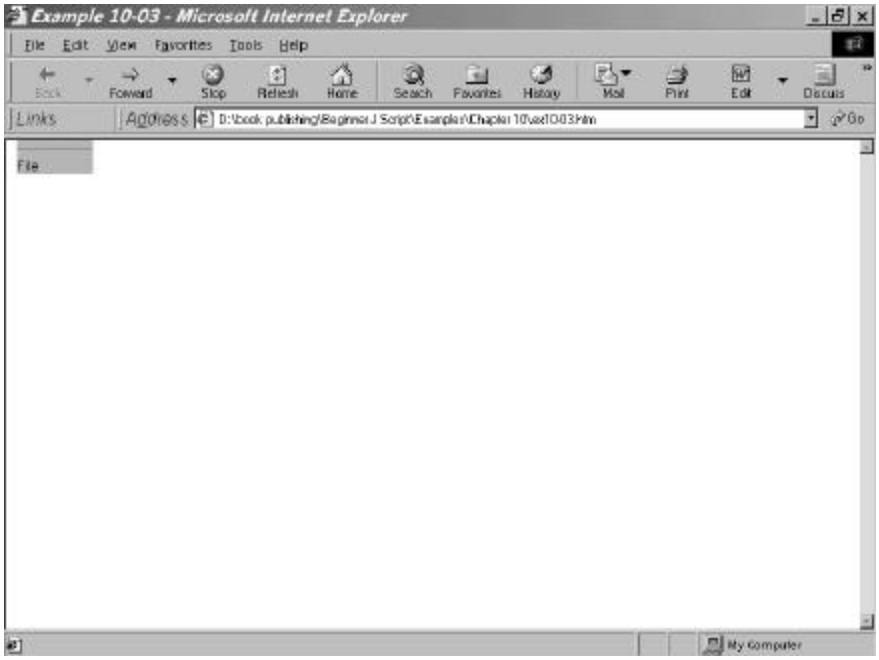


Fig ure 10-3

Fig ure 10-4

Now let's look at this exam ple in more detail to ensure that you have a firm under stand ing of exactly what is occur ring here.

The first part of the JavaScript code may seem a bit strange:

```
var browser
if (document.getElementById)
{
     browser = true
}
else
{
     browser =  false
}
```

What we are doing here is see ing if this browser will sup port the script we are writ ing. Since we use the doc u ment's ele ment object we see if this browser will sup port that, by sim ply call ing the getElementById func tion. If this works, then the browser will sup port our script. You will notice that in many scripts in this book, the first step is to make sure the browser will sup port the script. This is sim ply a good prac tice to get into, to keep vis i tors to your web site from get ting error mes sages.

The menu does most of its action in the toggleMenu func tion, so let's take a close look at that.

```
function toggleMenu(currElem,nextPos)
{
     menuObj = (browser ) ? document.getElementById(currElem).style :
               eval("document." + currElem)
     if (toggleMenu.arguments.length == 1)
     {
          nextPos = (parseInt(menuObj.top) == -5) ? -90 : -5
     }
     menuObj.top = (browser ) ? nextPos + "px" : nextPos
}
```

Don't let the appar ent com plex ity scare you; this is not as tough as it looks. The real new item here is what is called a *ter nary oper a tor*. This is an oper a tor that has three ele ments. Most oper a tors you have seen so far have one or two oper a tors (such as the unary incre ment oper a tor, + +, and the binary math oper a tors, +, –, *, /, etc.). The ter nary oper a tor has three argu ments. In the form we use it here, it is essen tially a con densed if-then state ment. We will look closely at the sec ond ter nary oper a tor in this script seg ment since it is shorter.

```
menuObj.top = (browser ) ? nextPos + "px" : nextPos
```

What is hap pen ing here is we are say ing that if the browser object is valid, then mneuObj.top should be set equal to nextPos + “px”, else sim ply set it to nextPos. You will see ter nary oper a tors used in many JavaScripts. They are also fre quently used in Java, C , and C + + pro gram ming.

What this func tion does, with the use of two ter nary oper a tors sim ply to shorten if-else state ments, is to set the posi tion of the menu. That is, to cause it to expand so you can view the full menu or to col lapse so that you can not.

To cre ate the menu we again use Cas cading Style Sheets. For our pur poses it is not impor tant to go into depth on the details of Cas cading Style Sheets. We can occa sion ally use them with out an in-depth knowl edge of them. We then sim ply use onMouseover to dis play the menu and onMouseout to make the menu dis appear.

Expanding Menus

Some times you have a lot of infor ma tion to dis play, but dis play ing it all makes for a very clut tered web page. Using an expand ing menu you can group infor-ma tion into top ics and then the user can click on a topic they are inter ested in and have the detailed infor ma tion dis play for that topic. Let’s look at an example:

Example 10-4

```
<HTML>
<HEAD>
<TITLE>Example 10-04</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT">
function toggleMenu(currMenu)
{
     if (document.getElementById)
     {
          thisMenu = document.getElementById(currMenu).style
               if (thisMenu.display == "block")
               {
                    thisMenu.display = "none"
               }
               else
               {
                    thisMenu.display = "block"
               }
                   return false
```

```
                }
                else
                {
                     return true
                }
}
</SCRIPT>
<STYLE TYPE="TEXT/CSS">
   .menu  {display:none; margin-left:20px}
</STYLE>
</HEAD>
<BODY BGCOLOR="WHITE">
<H3>
<A HREF="example10-04.htm" onClick="return toggleMenu('menu1')">Internet
Books from WordWare</A>
</H3>
<SPAN CLASS="menu" ID="menu1">
     Learn ASP in three days by Jose Ramalho<BR>
     Search Engine Positioning by Fredrick Marckini<BR>
     Learn Active X Scripting with MSIE by Nathan Wallace<BR>
</SPAN>
<H3>
<A HREF="example10-04.htm" onClick="return toggleMenu('menu2')">Game Design
books from WordWare</A>
</H3>
<SPAN CLASS="menu" ID="menu2">
Introduction to Computer Game Programming with Direct X  8.0 by Ian Parberry<BR>
Designing Arcade Computer Game Graphics by Ari Feldman<BR>
Real-Time Strategy Game Programming using Direct X by Mickey Kawick
</SPAN>
<H3>
<A HREF="example10-04.htm" onClick="return toggleMenu('menu3')">Database
Applications from WordWare</A>
</H3>
<SPAN CLASS="menu" ID="menu3">
     Learn Microsoft SQL Server 7.0 by Jose' Ramalho<BR>
     Learn Oracle 8i by Jose' Ramalho
</SPAN>
</BODY>
</HTML>
```

Again we have a script that is not nearly as com pli cated as it looks at first glance. Let's look this script over and see what is hap pen ing.

The first thing we do is use the getElementByID func tion of the doc u ment object to see if this script is even sup ported by the browser being used. As I

pre vi ously stated, it is good to get into the habit of check ing to see if the browser will sup port your script before attempt ing to exe cute the script. Once we have con firmed that the script can be sup ported, we sim ply change the menu style from its cur rent posi tion. If it is cur rently at "none," then we dis- play the block style menu. If it is cur rently at "block" then we change it to none. What this does is cause the menu to dis play the first time you click, and then to dis ap pear the sec ond time you click.

Once again we use some Cas cading Style Sheet tech niques to actu ally dis play the menu. How ever the call to our JavaScript func tion is done from a link. Since we don't actu ally want this link to go any where but to our script, I set <A HREF> to the page we are cur rently on.

Now if you typed in this exam ple prop erly, you should be able to open it in your browser and see some thing like the images shown here:

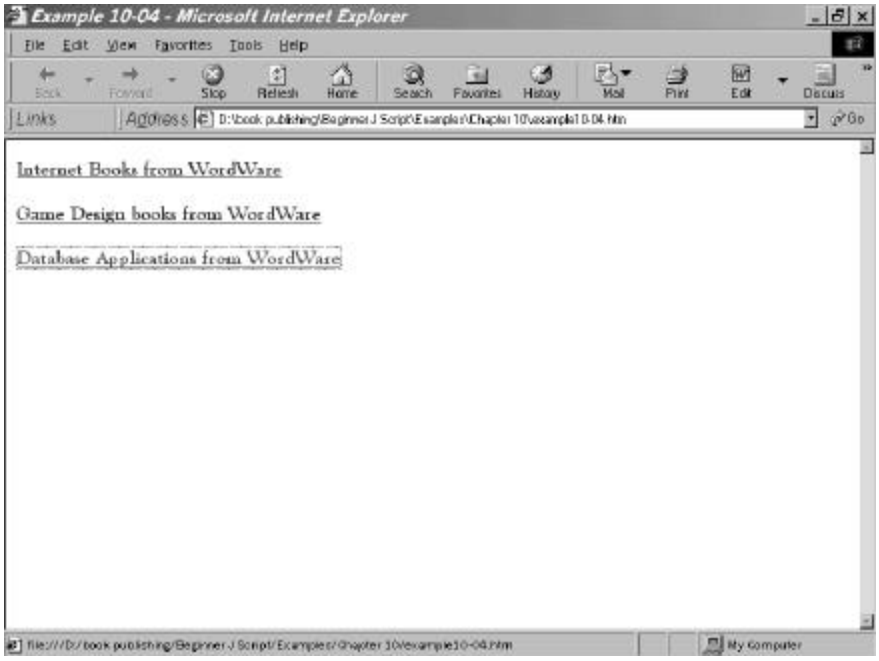


Fig ure 10-5

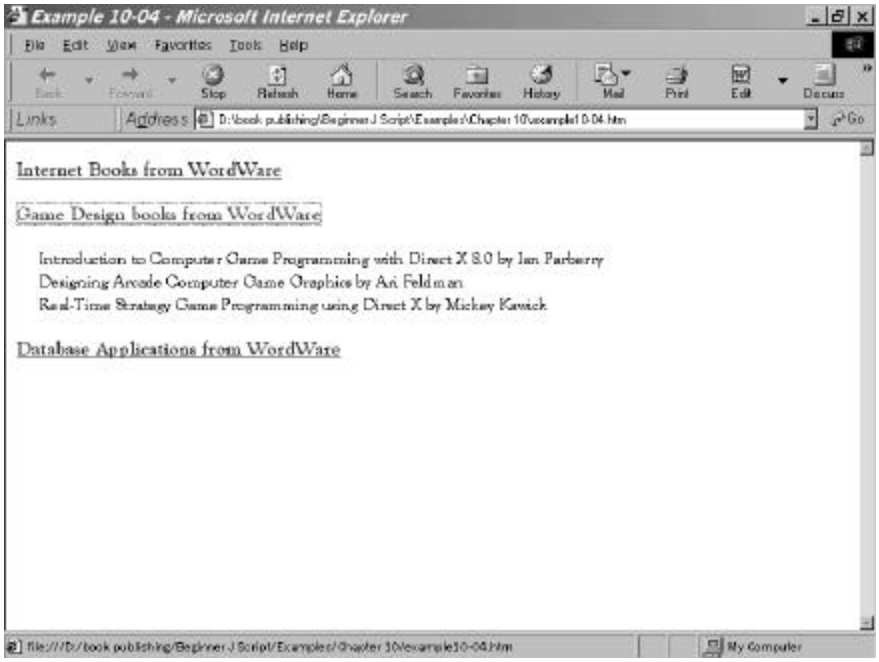


Figure 10-6

I think you will find that this particular type of menu is well suited for tasks such as displaying inventories. It allows you to initially display only categories and then the visitor to your web site can expand any category that he or she finds particularly interesting.

Antique Bookstore Project

As with all of the chapters in this book, we will now take the techniques we have covered and incorporate them into our ongoing antique book store project. If you choose to work through these project exercises at the end of each chapter, then by the time you finish the book you will have built an entire web site with many exciting features.

We are going to add a complete inventory page that uses the same menu style we implemented in Example 10-4.

```
<HTML>
<HEAD>
<TITLE>Complete Inventory</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT">

function toggleMenu(currMenu)
```

```
 {
        if (document.getElementById)
        {
             thisMenu = document.getElementById(currMenu).style
                 if (thisMenu.display == "block")
                 {
                      thisMenu.display = "none"
                 }
                 else
                 {
                      thisMenu.display = "block"
                 }
                 return false
        }
        else
        {
             return true
        }
}

</SCRIPT>
<STYLE TYPE="TEXT/CSS">
   .menu  {display:none; margin-left:20px}
</STYLE>
</HEAD>
<BODY BGCOLOR="WHITE">

<H3>
<A HREF="completeinventory.htm"onClick="return toggleMenu('menu1')">Books
from the 1800's</A>
</H3>
<SPAN CLASS="menu" ID="menu1">
     1st Edition Charles Dicken's Tale of Two Cities<BR>
     1st Edition Edgar Allen Poe's The Tell Tale Heart<BR>
     Signed Copy of Edgar Allen Poe's The Cask of Amontillado<BR>

</SPAN>
<H3>
<A HREF="completeinventory.htm" onClick="return toggleMenu('menu2')">Books
from the 1700's</A>
</H3>
<SPAN CLASS="menu" ID="menu2">
     Ben Franklins Memoirs<BR>
     How to start a revolution<BR>
     Political Theory by Thomas Paine
</SPAN>
```

```
<H3>
<A HREF="completeinventory.htm" onClick="return toggleMenu('menu3')">Books
for under $25</A>
</H3>
<SPAN CLASS="menu" ID="menu3">
     A cowboy's diary (circa 1850's)<BR>
     4th edition Charles Dicken's A tale of two cities
     3rd Edition Jules Verne 20000 Leagues under the sea
</SPAN>

<H3>
<A HREF="completeinventory.htm" onClick="return toggleMenu('menu4')">Our
finest items</A>
</H3>
<SPAN CLASS="menu" ID="menu4">
     Signed copy of Shakespeare's Hamlet<BR>
     Gutenberg's Diary<BR>
     Signed first Edition Longfellow
     1570 Hebrew Bible
</SPAN>
</BODY>
</HTML>
```

Summary

This chap ter intro duced you to the pro cess of add ing menus to your web pages. It gave you exam ples of four very dif fer ent types of menus. The goal was to show you how to inte grate JavaScript with other web design tech nol o-gies such as HTML forms and Cas cading Style Sheets to cre ate dynamic menus for your web site. You will find that if you use menus on your web sites, nav i ga tion is much eas ier for your web sites vis i tors, thus increas ing their sat-is fac tion with your web page.

Chapter 11

Working with Forms

Through out the past sev eral chap ters of this book you have worked with var i-ous ele ments of HTML forms. I have included these ele ments with out much expla na tion. This chap ter is devoted to explain ing exactly how HTML forms function.

Essen tially, HTML stan dards allow you to cre ate a vari ety of form ele ments that make user input much eas ier. So far in this book you have already seen the text field and the but ton used. In this chap ter we will be exam in ing the con cepts involved. We will also be look ing at other form ele ments you can use in HTML. It is a very com mon tech nique to call your JavaScript func tions from an HTML form event.

Form Basics

Since forms are a part of HTML they are defined by tags, like every thing else in HTML. The basic tag to define that you are going to use form ele ments is the <FORM> tag.

```
<FORM>
</FORM>
```

All of the form ele ments you wish to use must go between these two tags. Each form ele ment you might wish to use is defined in a very sim i lar way. You define the ele ment type (but ton, text field, etc.), its name (what to refer to it as in your code), and its ini tial value. In some cases you also define other prop-er ties such as size. For exam ple, if you wish to place a text field on your HTML doc u ment so your users can enter data it works like this:

Example 11-1

```
<HTML>
<HEAD>
```

```
    <TITLE>Example 11-01</TITLE>
</HEAD>
<BODY>
<FORM>
   <INPUT TYPE=text NAME="txttest" VALUE="Enter Text Here">
<FORM>
</BODY>
</HTML>
```

If you enter all the code cor rectly, you should be able to view the page in your browser and see some thing sim i lar to this:

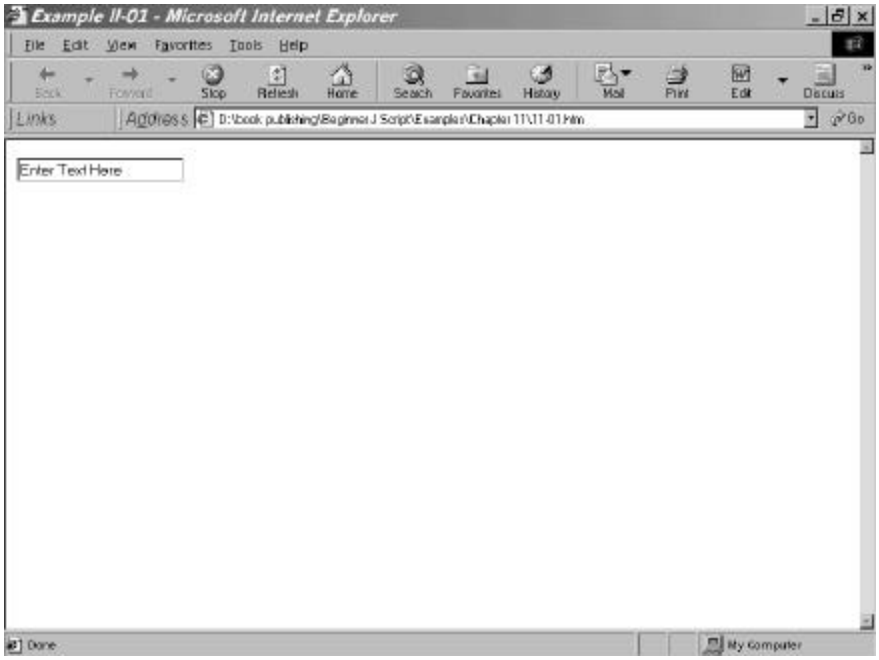


Fig ure 11-1

As you can see in this exam ple, inside the <FORM> and </FORM> tags we define the form ele ments we wish to dis play. The first thing we define is what type of form ele ment it will be. In this exam ple it is a text field. We then give it a name. We can use that name later in either HTML or in our JavaScript to refer to that form ele ment. Finally, we put an ini tial value in the text field; this is optional. As you can see, cre at ing form ele ments is rather sim ple.

Text Fields and Buttons

Pos si bly the two most com monly used form ele ments are the text field (which we saw in the pre vi ous exam ple) and the but ton. The but ton is an excel lent form ele ment to call your JavaScripts from. Let's look at an exam ple that does just that.

Example 11-2

```
<HTML>
<HEAD>
    <TITLE>Example 11-02</TITLE>
    <SCRIPT LANGUAGE = "JavaScript">
              alert(txttest.value)
    </SCRIPT>
</HEAD>
<BODY>
<CENTER>
  <INPUT TYPE=text NAME="txttest" VALUE="Enter Text Here"><BR>
  <INPUT TYPE=button NAME="Submit" VALUE="Submit" onclick="test()">
</CENTER>
</BODY>
</HTML>
```

This exam ple shows a minor but impor tant twist on the usage of form elements. Here we have a but ton, and when it is clicked (onClick event) we call a JavaScript func tion. This func tion, in turn, ref er ences the value in the HTML form's text field. If you enter all the code prop erly you will be able to see some thing very sim i lar to the fol low ing fig ure:

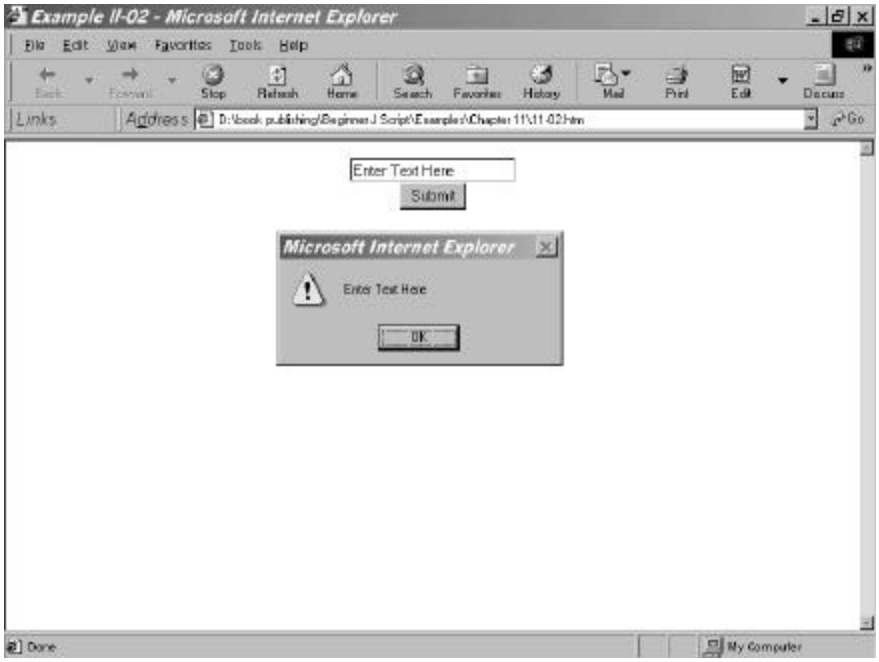


Fig ure 11-2

Using but tons to trig ger JavaScript func tions is both quite com mon and very use ful. This tech nique allows you to call func tions in response to user activ ity. In our exam ple, the JavaScript we call is quite sim ple, but it illus trates the point quite well.

Options

An option in HTML is the same thing as a combo box in Visual Basic. It is a drop-down box that dis plays a list of choices the user can select from. This can be quite use ful if you wish the user to select from mul ti ple items but do not wish to clut ter your screen. Con sider the fol low ing exam ple:

Example 11-3

```
<HTML>
<HEAD>
    <TITLE> Example 11-03</TITLE>
<SCRIPT>
function moveon()
{
window.location=document.dropdown.selection.options[document.dropdown.selection.
selectedIndex].value
}
```

```
</SCRIPT>
<BODY BGCOLOR = "White">
<FORM NAME="dropdown">
          <SELECT NAME="selection"
            <OPTION VALUE=""><B>Select an Option</B>
            <OPTION VALUE="this.html">this one
            <OPTION VALUE="next.html">next one
            <OPTION VALUE="after.html">one after
          </SELECT>
          <INPUT TYPE=button NAME="Submit" VALUE="Submit" onclick="moveon()">
</FORM>
</BODY>
</HTML>
```

This exam ple sim ply allows the user to select which web page they would like to nav i gate to. Once they have selected the page, they press the Sub mit but ton and are taken to that page. The script is actu ally only one line. It sim ply uses the win dow.loca tion prop erty to move to what ever web page the user selected. If you enter the code prop erly, you should be able to see some thing like this:

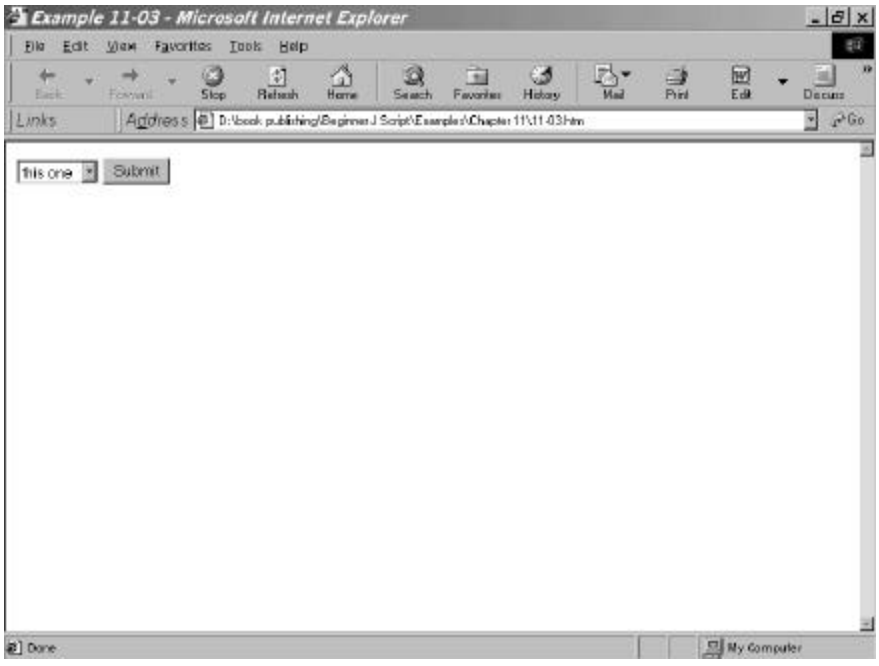


Fig ure 11-3

Radio Buttons

Radio but tons are the small cir cles that you click in to select a par tic u lar option. You have seen these before in web pages and on desk top appli ca tions. Radio but tons pres ent a series of choices to the user, the user may select only one. Look at this exam ple:

Example 11-4

```
<HTML>
<HEAD>
    <TITLE>Example 11-04</TITLE>

<SCRIPT LANGUAGE="JavaScript">
</SCRIPT>
</HEAD>
<BODY>
<FORM>
  <INPUT TYPE="radio" NAME="radiobutton" onClick="radio_selection='A'">Add
  <INPUT TYPE="radio" NAME="radiobutton" onClick="radio_selection='D'">Delete
  <INPUT TYPE="radio" NAME="radiobutton" onClick="radio_selection='U'">Update
</FORM>
</HTML>
```

As you can see, we sim ply use INPUT TYPE = “radio” to indi cate that this is going to be an input type, and the input will be done via a radio but ton. We then give the radio but tons a name. Notice they all have the same name. Since the user can only select one but ton, we are sim ply inter ested in set ting the value based on one but ton.

If you enter the code exactly as shown in the exam ple, you will then be able to see the fol low ing screen:

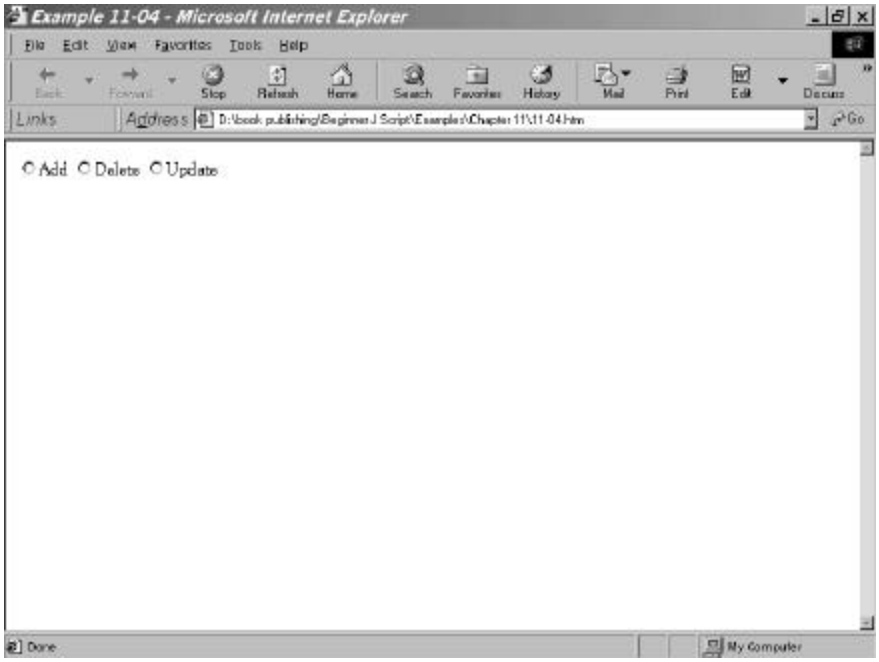


Figure 11-4

Check Boxes

Radio buttons present several choices but only allow you to select one. This is very useful in many cases, such as if you are asking the user's marital status: married, single, or divorced. They can only be one at any given time. However, with other types of information a person might be able to select more than one choice. For example, if you ask a person to indicate which sports they watch on TV (football, basketball, baseball, golf), they may want to choose more than one. For this type of data, you should use a checkbox. A check box is similar to an option box, but the person can select multiple options. The following example illustrates this.

Example 11-5

```
<HTML>
<HEAD>
    <TITLE>Example 11-05</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var allchecked = "false"
function check(field)
{
     if (allchecked  == "false")
     {
```

```
            for (i = 0; i < field.length; i++)
            {
                 field[i].checked = true;
            }
            allchecked  = "true"
            return "Uncheck All"
       }
      else
      {
           for (i = 0; i < field.length; i++)
           {
                field[i].checked = false
           }
      allchecked  = "false"
      return "Check All"
      }
}

</SCRIPT>
</HEAD>
<BODY BGCOLOR = White>
<FORM NAME =myform>
<b>JavaScript Websites you like!</b><br>
<INPUT TYPE=checkbox name=list value="1">JavaScript Source<BR>
<INPUT TYPE=checkbox name=list value="2">JavaScript.Com<BR>
<INPUT TYPE=checkbox name=list value="3">JavaScript World<BR>
<INPUT TYPE=checkbox name=list value="4">HTML Goodies <BR>
<BR>
<INPUT TYPE=button VALUE="Check All" onClick="this.value=check(this.form.list)">
</FORM>
</HTML>
```

In this exam ple we use the <FORM> tag to indi cate that form ele ments are going to be used. We then use INPUT TYPE = checkbox. This lets the browser know it should dis play a checkbox. If you enter all the code prop erly, you should see some thing like the fol low ing screen.

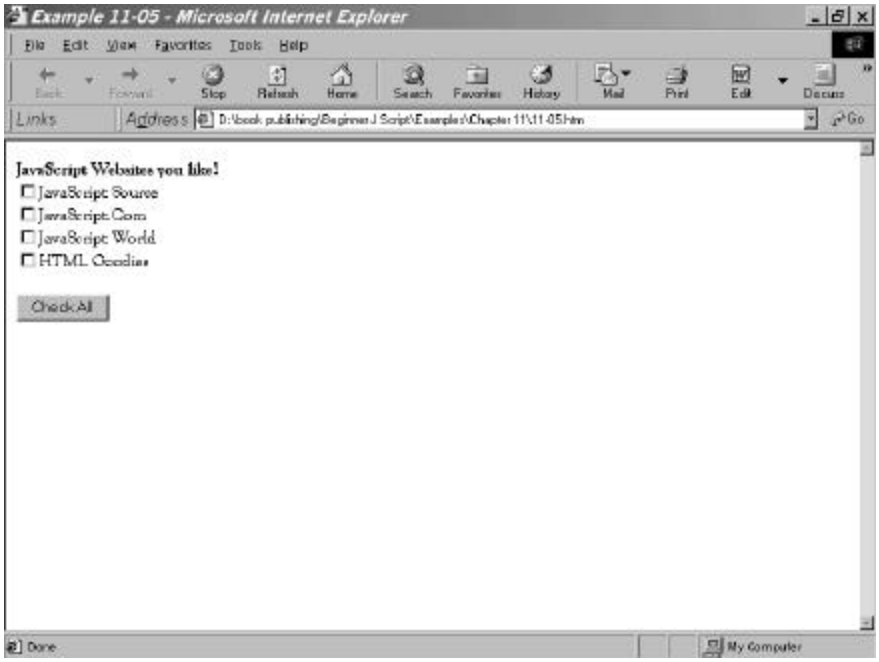


Figure 11-5

The checkbox can be very useful in allowing the user to select multiple options.

Event Handlers in Form Elements

There are a variety of event handlers that allow you to invoke specific JavaScript code when the user performs a given action. We will examine a few of the most common here.

onSubmit

The onSubmit event handler is an attribute of the <FORM> tag. It is called when a form is submitted. A form can be submitted in several ways; the Submit button is only one of many. The submit event can occur immediately upon clicking a submit button, pressing Enter, or several other methods. JavaScript triggers this event prior to sending the data to the server. The event handler's script is executed before the form's data is actually submitted to the server for further processing.

The onSubmit event han dler is a com mon place used to val i date the con tent of a form's ele ment. Cli ent-side form val i da tion is gain ing pop u lar ity because the user receives an imme di ate response regard ing invalid entries. For exam ple, if you have a form with a text box in which the user is asked to type his or her e-mail address, you can use a sim ple JavaScript script that will make sure (upon sub mis sion) the user's entry is a string con tain ing an "at" sign (@), which, of course, all valid email address will have.

You can use the onSubmit event han dler not just to val i date the form's elements but also to can cel its sub mis sion alto gether. The form's sub mis sion is aborted when the event han dler returns a false value, as in the fol low ing example:

```
<FORM NAME="form1" onSubmit="return false">
```

Of course, this exam ple is not very use ful because it dis ables the form sub mission uncon di tion ally. How ever, it does illus trate the point in ques tion. Usually, a func tion val i dates the form and returns a true or false value accord ingly. You can use the fol low ing code to can cel or pro ceed with the form sub mis sion, depend ing on the value returned by the func tion:

```
<FORM NAME="form1" onSubmit="return checkData()">
```

The fol low ing exam ple illus trates these con cepts with a form con tain ing a text area box and a Sub mit but ton which e-mails you the con tents of the text area after prompt ing the user for con fir ma tion:

Example 11-6

```
<SCRIPT LANGUAGE="JavaScript">
function continue()
 {
   return confirm("Click OK to mail this information")
}

</SCRIPT>
<FORM ACTION="mailto:chuckeasttom@yahoo.com" METHOD="post" ENCTYPE=
              "text/plain" onSubmit="return continue()">
       <INPUT TYPE =textfield NAME="txttest" Value = "Put Message Here"><BR>
       <INPUT TYPE=button VALUE="Send Mail">
</FORM>
```

If you enter all the code cor rectly, you should be able to use your browser and see the fol low ing image:

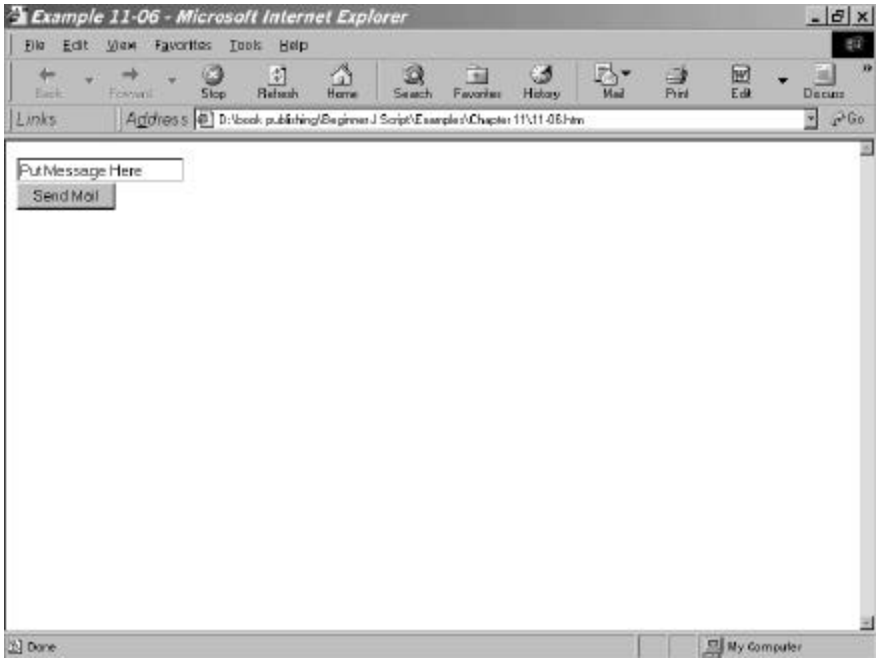


Figure 11-6

Some of the parameters used here you have seen before, but ENCTYPE may be new to you. In order to receive the form's content as plain, unscrambled e-mail, you need to assign a "text/plain" value to the ENCTYPE attribute.

onReset

Another event handler of the <FORM> tag is onReset. A reset event usually occurs when the user clicks a Reset button. The onReset event handler behaves very much like the onSubmit event handler.

The following example asks the user to confirm the resetting process before executing it:

Example 11-7

```
<HTML>
<HEAD>
    <TITLE>Example 11-07</TITLE>
</HEAD>
<BODY BGCOLOR = White>
<FORM ACTION="mailto:yshiran@iil.intel.com" METHOD="post" ENCTYPE="text/plain"
onReset="return confirm('Click OK to reset form to default status')">
<TEXTAREA NAME="input" COLS=40 ROWS=10></TEXTAREA><BR>
<INPUT TYPE="reset" VALUE="reset it!">
```

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

In this exam ple, if you press the Reset but ton, you are prompted to con firm the action. Then all the text is reset to its default value, which hap pens to be blank. If you enter all the code cor rectly, you should be able to use your browser and see the fol low ing image:

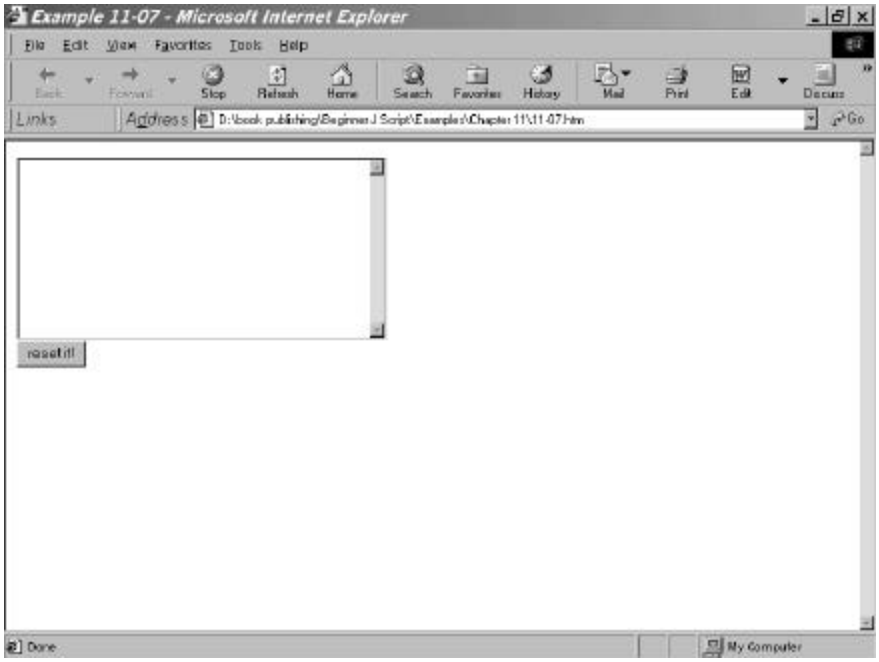


Fig ure 11-7

Synopsis of Form Elements

The fol low ing table sum ma rizes the major form ele ments avail able to you in HTML.

HTML Element	Value of Type Attribute
INPUT TYPE="button"	"button"
INPUT TYPE="checkbox"	"checkbox"
INPUT TYPE="file"	"file"
INPUT TYPE="hidden"	"hidden"
INPUT TYPE="password"	"password"
INPUT TYPE="radio"	"radio"
INPUT TYPE="reset"	"reset"
INPUT TYPE="submit"	"submit"

HTML Element	Value of Type Attribute
INPUT TYPE="text"	"text"
SELECT	"select-one"

Antique Bookstore Project

This chapter is one of the rare ones where we do not add a new item to our antique bookstore project. This is because we have already used form elements in previous chapters. This chapter's goal was simply to clarify form issues.

Summary

Throughout this book we have used form elements. Prior to this chapter you have simply typed them in and may not have realized exactly what you were doing. After studying this chapter you should have a good working understanding of how HTML forms work.

Chapter 12

Strings in JavaScript

Strings have many uses in your scripts. Strings hold text data, such as names, addresses, etc. You can store string val ues in a stan dard vari able. How ever, if you cre ate a string object to store your string data, you then can use the var i-ous string object meth ods to manip u late that string. Essen tially, strings are sim ply an array of indi vid ual char ac ters. The index ing of strings, like that of arrays, is zero-based. The index of the first char ac ter of a string is 0, the index of the sec ond one is 1, that of the third one is 2, and so on.

Creating Strings

String is a built-in object. This means that it is han dled a bit dif fer ently than other vari ables. It is cre ated like an object vari able, rather than like a sim ple vari able. Here is an exam ple:

```
var lastname = new String("Smith")
```

The gen eral syn tax is:

```
Var name = new String(string)
```

Name is the name of the string object you are cre at ing. You can give it any name you wish, but I rec om mend a name that reflects the type of data it will hold. *string* is any lit eral string, as in the exam ple.

String Length

The String object com bines many use ful meth ods and one prop erty, length. This prop erty reflects the length of its call ing object. Here is an exam ple:

Example 12-1

```
<HTML>
<HEAD>
    <TITLE>Example 12-01</TITLE>
```

```
    <SCRIPT LANGUAGE = "JavaScript">
        function test()
        {
             var mystr =txttest.value
             alert(mystr.length)

         }
    </SCRIPT>
</HEAD>
<BODY>
<CENTER>
  <INPUT TYPE=text NAME="txttest" VALUE="Enter Text Here"><BR>
  <INPUT TYPE=button NAME="Submit" VALUE="Submit" onclick="test()">
</CENTER>
</BODY>
</HTML>
```

If you enter the exam ple prop erly, you can use your browser to view it and you should see the fol low ing:

Fig ure 12-1

String Methods

As men tioned, the String object has many use ful meth ods, one of which is the toUpperCase() method. This changes all the text in the string to upper case. There are many prac ti cal appli ca tions for this method.

CharAt()

The charAt() method returns the char ac ter whose index is equal to the argument of the method. The char ac ters of a string are indexed from 0 to length–1. The gen eral syn tax is:

```
mystring.charAt(index)
```

Here is an exam ple:

Example 12-2

```
<HTML>
<HEAD>
    <TITLE>Example 12-02</TITLE>
    <SCRIPT LANGUAGE = "JavaScript">
        function test()
        {
             var mystr =txttest.value
             alert(mystr.charAt(1))

        }
    </SCRIPT>
</HEAD>

<BODY>
<CENTER>
  <INPUT TYPE=text NAME="txttest" VALUE="Enter Text Here"><BR>
  <INPUT TYPE=button NAME="Submit" VALUE="Submit" onclick="test()">
</CENTER>
</BODY>
</HTML>
```

If you entered the code prop erly, and then entered some text in the box you should see some thing sim i lar to the fol low ing fig ure:

Fig ure 12-2

The fol low ing script seg ment prints the char ac ter “o,” because it is the sec ond char ac ter (index 1) in the string. You can also call this method with a lit eral text as in the fol low ing exam ple:

```
document.write("wordware".charAt(1))
```

You can print the char ac ters of a string via a sim ple loop:

```
var mystring = "Wordware Publishing”
for (var i = 0; i < mystring.length; ++i)
{
     document.write(mystring.charAt(i))
}
```

At first, a string lit eral is assigned to the vari able mystring. The loop then iterates length times. It starts at 0, and ends at mystring.length – 1.

indexOf()

This method returns the index of the first occur rence of the spec i fied substring in the call ing string object, start ing the search at the begin ning of the string. An exam ple will surely clear things up:

```
var mystring = "wordware"
document.write(str.indexOf("wo"))
```

This script's output is the number 0. The first occurrence of the substring "wo" in the calling string object is at the second character whose index is 0. The search for the specified substring starts at index 0, the beginning of the string.

lastIndexOf()

This method is identical to the indexOf method, except that it returns the index of the last occurrence of the specified value, rather than the first occurrence. Its syntax is, obviously, the same:

```
stringName.lastIndexOf(searchValue, [fromIndex])
```

The following script prints the number 3:

```
var mystring = "a/b/c"
document.write(str.lastIndexOf("/"))
```

substring()

Strings are constructed of characters. The substring() method returns a set of characters within its calling String object. Its general syntax is:

```
stringName.substring(indexA, indexB)
```

stringName is any string. *indexA* and *indexB* are both integers between 0 and *stringName*.length – 1. *indexA* is the index of the first character in the substring, whereas *indexB* is the index of the last character in the substring plus 1. Consider the following example:

Example 12-3

```
<HTML>
<HEAD>
    <TITLE>Example 12-03</TITLE>
    <SCRIPT LANGUAGE = "JavaScript">
        function test()
        {
             var begin = txtbegin.value
             var end = txtend.value
             var mystring =txttest.value
             var seg = mystring.substring(begin,end)
             alert(seg)

        }
    </SCRIPT>
```

```
</HEAD>

<BODY>
<CENTER>
  <INPUT TYPE=text NAME="txttest" VALUE="Enter Text Here"><BR>
  <INPUT TYPE=text NAME="txtbegin" VALUE="0">
  <INPUT TYPE=text NAME="txtend" VALUE="5"><BR>
  <INPUT TYPE=button NAME="Submit" VALUE="Submit" onclick="test()">
</CENTER>
</BODY>
</HTML>
```

If you entered every thing prop erly, you should see some thing like the fol lowing in your browser:

Fig ure 12-3

Notice that the length of the substring is *indexA – indexB*.

Number-to-String Conversion

Occa sionally, you need to con vert a num ber to a string. For exam ple, if you want to com pute the num ber of dig its in a num ber, you can con vert it to a string and use the length prop erty, which applies to strings only. In this sec tion we shall take a look at a few ways to con vert a num ber into a string.

Empty String Concatenation

The easiest way to convert a number to a string is by concatenating an empty string to the number. Here is an example:

```
var num = 2001
num = num + ""
```

You can also convert the number to a string and assign the numeric string to another variable, or, even better, do both operations in one statement:

```
var num = 2001
var numericString = num + ""
```

Antique Bookstore Project

Most of the chapters in this book end by adding something to our ongoing antique bookstore project. However, this chapter is more about some useful techniques for string manipulation and is not easily suited to our ongoing project. For this reason, this chapter will not be adding to our project. However, don't worry, the next chapter will add plenty!

Summary

While you can store string data in a standard variable, I hope you have seen that the String object can be quite useful. You will find many practical situations where you will need to know a string's length or other properties. You may also need to periodically manipulate the contents of a string. With JavaScript's String object you can do that.

Chapter 13

Math in JavaScript

We have been able to do a lot of inter est ing things with out using any math. How ever, at some point you will prob a bly need to use math in your JavaScript. This is espe cially true for busi ness websites.

Math in JavaScript is basi cally divided into two areas:

- Mathematicaloperators
- The built-in Math object

Mathematical Operators

You are prob a bly famil iar with most math oper a tors. They were intro duced to you in Chap ter 2. But just in case you need a bit of a refresher, the fol low ing table should help you.

Operator	Symbol	Explanation	Example
Addition	+	Adds the numbers on either side together	Var mynum Mynum = mynum + 1
Subtraction	–	Subtracts the number on the right from the number on the left	Var mynum = mynum – 1
Multiplication	*	Multiplies the numbers on either side together	Var mynum = mynum * 5
Division	/	Divides the number on the left-hand side by the number on the right-hand side	Var mynumMynum = mynum /4
Modulus	%	This operator provides you with the remainder of a division operation.	Var answerAnswer = 9%4(answer will be 1)
Increment	+ +	Increments the number by one	Var mynumMynum+ +
Decrement	—	Decreases the number by one	Var mynumMynum—

Here is a piece of code that uses these operators in conjunction with form elements to create a simple calculator.

Example 13-1

```
<HTML>
  <HEAD>
    <TITLE>Example 13-01</TITLE>
 </HEAD>
<BODY>
<CENTER>
<FORM NAME="calculator">
<TABLE BORDER=1 BGCOLOR = green>
<TR>
<TD>
<INPUT TYPE="text"   NAME="Input" Size="16">
<BR>
  <TR>
     <TD>
         <INPUT TYPE="button" NAME="one"   VALUE="  1  "
                    OnClick="calculator.Input.value += '1'">
        <INPUT TYPE="button" NAME="two"   VALUE="  2  "
                    OnCLick="calculator.Input.value += '2'">
        <INPUT TYPE="button" NAME="three" VALUE="  3  "
                    OnClick="calculator.Input.value += '3'">
       <INPUT TYPE="button" NAME="plus"  VALUE="  +  "
                    OnClick="calculator.Input.value += ' + '">
  <TR>
     <TD>
        <INPUT TYPE="button" NAME="four"  VALUE="  4  "
                    OnClick="calculator.Input.value += '4'">
       <INPUT TYPE="button" NAME="five"  VALUE="  5  "
                    OnCLick="calculator.Input.value += '5'">
       <INPUT TYPE="button" NAME="six"   VALUE="  6  "
                    OnClick="calculator.Input.value += '6'">
       <INPUT TYPE="button" NAME="minus" VALUE="  -  "
                    OnClick="calculator.Input.value += ' - '">
  <TR>
     <TD>
      <INPUT TYPE="button" NAME="seven" VALUE="  7  "
                    OnClick="calculator.Input.value += '7'">
     <INPUT TYPE="button" NAME="eight" VALUE="  8  "
                    OnCLick="calculator.Input.value += '8'">
     <INPUT TYPE="button" NAME="nine"  VALUE="  9  "
                    OnClick="calculator.Input.value += '9'">
     <INPUT TYPE="button" NAME="times" VALUE="  x  "
                    OnClick="calculator.Input.value += ' * '">
```

```
  <TR>
     <TD>
     <INPUT TYPE="button" NAME="clear" VALUE="  c  "
                    OnClick="calculator.Input.value = ''">
    <INPUT TYPE="button" NAME="zero"  VALUE="  0  "
                    OnClick="calculator.Input.value += '0'">
    <INPUT TYPE="button" NAME="DoIt"  VALUE="  =  "
                    OnClick="calculator.Input.value = eval(Calc.Input.value)">
    <INPUT TYPE="button" NAME="div"   VALUE="  /  "
                    OnClick="calculator.Input.value += ' / '">
<BR>

</TABLE>
</FORM>
</CENTER>
</BODY>
</HTML>
```

The Math Object

In the pre vi ous chap ter you were intro duced to the JavaScript String object. In this chap ter I will intro duce you to the JavaScript Math object. This object has a vari ety of math e mat i cal func tions that you may find quite use ful in your scripts.

To access ele ments of the Math object, you do not need to cre ate an instance of the Math object. For exam ple, the PI con stant is a prop erty of the Math object, and can be accessed via the fol low ing syn tax:

```
var pi = Math.PI
```

Notice you did not have to estab lish a vari able for the Math object itself.

Constants

There are a num ber of math con stants built into the Math object. These constants are sim ply math e mat i cal con stants that are fre quently used. Some of them, at least, should be famil iar to you. If any are unfa mil iar, then sim ply skip them. If you don't know what a par tic u lar math e mat i cal con stant is, chances are you won't need to use it in your scripts.

E

A very important constant in mathematics is Euler's constant. Its approximate value is 2.718281828459045. It is rounded off to 15 digits after the decimal point.

In JavaScript you refer to it via a capital E; that is, Math.E. (In mathematics it is usually referred to with a lower case e.)

LN2

Another constant featured as a property of the Math object is the natural logarithm of 2. Its approximate value is 0.6931471805599453.

JavaScript refers to this number as LN2. Because it is a property of the Math object, you should specify them together, as in Math.LN2.

You can use the pow method to assure that the preceding equation is true:

```
document.write(Math.pow(Math.E, Math.LN2))
```

Because both Euler's constant and the natural logarithm of 2 are approximate, the output of this statement is also an approximate:

```
1.9999999999999998
```

LN10

The natural logarithm of 10 is also featured as a property of the static Math object. Its value, as stored in its corresponding property, is 2.302585092994046.

In JavaScript this value is referred to as Math.LN10.

Here is a JavaScript statement to define the natural logarithm of 10:

```
document.write(Math.pow(Math.E, Math.LN10))
```

Since both Euler's constant and the natural logarithm of 10 are approximate, the output of this statement is also approximate:

```
10.000000000000002
```

LOG2E

Another important constant in the math arena is the base-2 logarithm of Euler's constant. Its approximate value is 1.4426950408889634. In math that is:

2[illegible]e

As you can see, you refer to this con stant in JavaScript as Math.LOG2E. Here is a sim ple state ment to con firm the value:

```
document.write(Math.pow(2, Math.LOG2E) - Math.E)
```

This time the out put is appar ently exact:

```
0
```

LOG10E

The base-10 log a rithm is also widely used in com plex math e mat i cal cal cu la-tions. Its value in JavaScript is approx i mately 0.4342944819032518.

As you can see, the equa tion is built accord ing to one of the basic log a rithm rules. In JavaScript, log base-10 of Euler's con stant is a prop erty of the Math object: Math.LOG10E.

Here is a sim ple script for con fir ma tion:

```
document.write(Math.pow(10, Math.LOG10E) - Math.E)
```

Once again, the out put is exact:

```
0
```

PI

Prob a bly the most known value among all con stants fea tured by JavaScript is PI. PI is used in many equa tions involv ing cal cu la tions with cir cles. Its approx-i mate value is 3.14.

SQRT2

The square root of 2 is also a well-known con stant. Its approx i mate value is 1.4142135623730951. You refer to it as Math.SQRT2. You can use the fol low-ing state ment to ensure the value:

```
document.write(Math.pow(Math.SQRT2, 2))
```

As you could expect, the result is not an exact one:

```
2.0000000000000004
```

Math Methods

Con stants are inter est ing, but for most busi ness appli ca tions you won't need to use any of these math e mat i cal con stants. On the other hand, the meth ods of the Math object will prob a bly be very use ful to you.

The meth ods of the Math object can be divided into two cat e go ries:

- Arithmeticmethods
- Trigonometricmethods

Arithmetic Methods

Let's start by exam in ing the arith me tic meth ods since they are used more frequently.

abs()

You can cal cu late the abso lute value of any num ber by using this method. The abso lute value of a num ber is its dis tance from zero.

For exam ple, the abso lute value of –5 is 5. The abso lute value of 5 is also 5. In JavaScript you can cal cu late the abso lute value of a num ber via the method Math.abs(). This method returns the abso lute value of its argu ment.

ceil()

The Math.ceil() method accepts a sin gle numeric argu ment and returns the next inte ger greater than or equal to the argu ment (round ing up). There fore, the returned value is never less than the argu ment. Here are a few exam ples:

```
Math.ceil(16) == 16
Math.ceil(16.01) == 17
Math.ceil(-15.01) == -15
```

Now if you are won der ing when you would ever use this func tion, don't worry, I have a prac ti cal exam ple for you. Let's say you need to fit a cer tain num ber of peo ple into a con fer ence room where each table can only seat four peo ple. You can use the Math.ceil() method along with a func tion to cal cu late the min i-mum num ber of tables:

```
function getNumLots(numpeople)
{
     return Math.ceil(numpeople / 4)
}
```

Since you can not use a part of a table, this method helps us find out how many tables we need.

floor()

The Math.floor() method is vir tu ally iden ti cal to the ceil ing method except that it returns the great est inte ger less than or equal to the value passed to it. Here are a few exam ples:

```
Math.floor(16) == 16
Math.floor(16.01) == 16
Math.floor(-15.01) == -16
```

log()

This method sim ply returns the nat u ral log a rithm of the argu ment passed to it.

max(), min()

Both of these meth ods accept two numeric argu ments. max() returns the greater of two num bers, whereas min() returns the lesser of the two. Here is a func tion that prints the lesser of two num bers fol lowed by the greater:

```
function printInOrder(num1, num2)
{
     document.write(Math.min(num1, num2) + ", " + Math.max(num1, num2))
}
```

The fol low ing func tion call prints the string "–5, 1" to the doc u ment:

```
printInOrder(1, -5)
```

Here are a few true expres sions to dem on strate the basic min() and max() methods:

```
Math.max(1, 2) == 2
Math.min(2, Math.abs(-2)) == 2
Math.min(2, -2) == -2
```

pow()

Given two numeric argu ments, this method returns the first one to the power of the sec ond. Here are a few true expres sions dem on strat ing the method:

```
Math.pow(10, 2) == 100
Math.pow(5,3) == 125
```

round()

The Math.round() method rounds a num ber to the near est inte ger. If the argument's dec i mal part is equal to 0.5, the num ber is rounded upward. Here are a few :

```
Math.round(3.7) == 4
Math.round(4.5) == 5
Math.round(16.1) == 16
Math.round(0) == 0
```

sqrt()

This method returns the square root of the argu ment. For exam ple:

```
Math.sqrt(4) == 2
Math.sqrt(0) == 0
Math.sqrt(0.25) == 0.5
```

If the argu ment is a neg a tive num ber, the method returns zero, which hap pens to be the wrong answer. JavaScript is not able to return imag i nary num bers (which are the square roots of neg a tive num bers).

Trigonometric Methods

Trig o no met ric meth ods are obvi ously those that deal with trig o nom e try.

Now if you don't know any thing about trig o nom e try, don't worry about it. You can be quite a suc cess ful JavaScript pro gram mer with out using trig o nom e try.

All angles in JavaScript are mea sured by radi ans rather than degrees because that is the stan dard used in trig o nom e try.

cos()

The Math.cos() takes just one argu ment, the angle of a tri an gle. It returns the cosine of that value, which we know must be spec i fied in radi ans. Here is an exam ple that prints out –1.

```
document.write(Math.cos(Math.PI))
```

The other trig o no met ric meth ods—sin, acos, cos, tan, atan—all work the same way. Here is an exam ple that com putes each of the trig o no met ric functions for a given angle:

Example 13-2

```
<HTML>
<HEAD>
    <TITLE>Example 13-02</TITLE>
    <SCRIPT LANGUAGE = "JavaScript">
        function trig()
        {
            var temp
            var angle
            angle = txttest.value

            temp = Math.sin(angle)
            txtsin.value = temp
```

```
            temp = Math.cos(angle)
            txtcos.value = temp

            temp = Math.tan(angle)
            txttan.value = temp

            temp = Math.asin(angle)
            txtasin.value = temp

            temp = Math.acos(angle)
            txtacos.value = temp

            temp = Math.atan(angle)
            txtatan.value = temp
         }
    </SCRIPT>
</HEAD>

<BODY>
<TABLE BORDER = 1 BGCOLOR = GRAY >
   <TR>
      <TD>
           Angle <INPUT TYPE=text NAME="txttest" VALUE=""><BR>
           <INPUT TYPE=button NAME="Submit" VALUE="Compute" onclick="trig()">
      <TD>
           <INPUT TYPE=text NAME="txtsin" VALUE=""> Sine  <BR>
           <INPUT TYPE=text NAME="txtcos" VALUE=""> Cosine<BR>
           <INPUT TYPE=text NAME="txttan" VALUE=""> Tangent<BR>
           <INPUT TYPE=text NAME="txtasin" VALUE=""> Arc sine  <BR>
           <INPUT TYPE=text NAME="txtacos" VALUE="">  Arc cosine <BR>
           <INPUT TYPE=text NAME="txtatan" VALUE="">Arc tangent<BR>
</BODY>
</HTML>
```

If you enter in all the code prop erly, you should be able to open your browser and see an image much like the fol low ing:

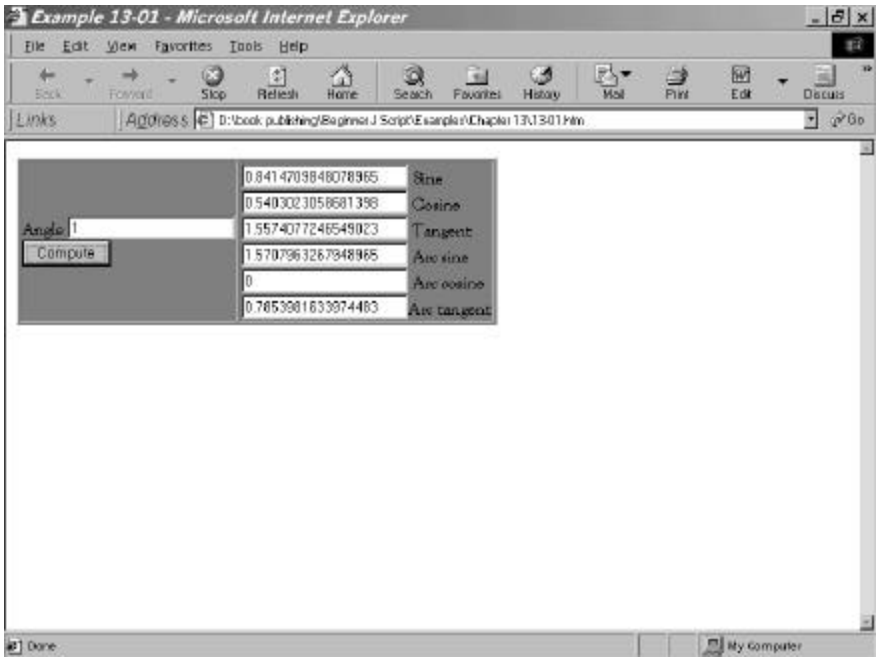


Fig ure 13-1

Antique Bookstore Project

As with most of the chap ters in this book we are going to use some of the tech niques pre sented in our antique book store pro ject. Obvi ously the trig o-nom e try func tions and many of the con stants are not going to be of par tic u lar use for an antique book store. How ever, the basic math oper a tors will be. We will add a screen that allows users to com pute sales tax and ship ping costs for books.

```
<HTML>
<HEAD>
   <TITLE>Computer Costs</TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT" >
  function total()
  {
   var shipping
   var tax
   var total

    var numbooks = txtbooks.value
    var price = txtprice.value
```

```
    tax = price * .0875
    shipping = numbooks * 1.5
    total = eval(price) + eval(tax) + eval(shipping)

    txttax.value = tax
    txtship.value = shipping
    txttotal.value = total
  }


</SCRIPT
</HEAD>
<BODY   background="back1.gif" >
<P>
<CENTER>
<TABLE BORDER = 1>
   <TR>
        <TD>
             Number of books ordered <INPUT TYPE=text NAME="txtbooks"
                   VALUE=""><BR>
             Total price of books ordered<INPUT TYPE = text NAME = "txtprice"
                   VALUE = ""><BR>
             <INPUT TYPE=button NAME="Submit" VALUE="Compute" onclick="total()">

   <TR>
       <TD>
             <INPUT TYPE=text NAME="txttax" VALUE=""> Tax<BR>
             <INPUT TYPE=text NAME="txtship" VALUE="">Shipping Costs <BR>
             <INPUT TYPE=text NAME="txttotal" VALUE=""> Total<BR>
   </TABLE>
</CENTER>
</BODY>
</HTML>
```

If you enter all the code cor rectly, you can use your browser to view the page and see some thing like the fol low ing screen.

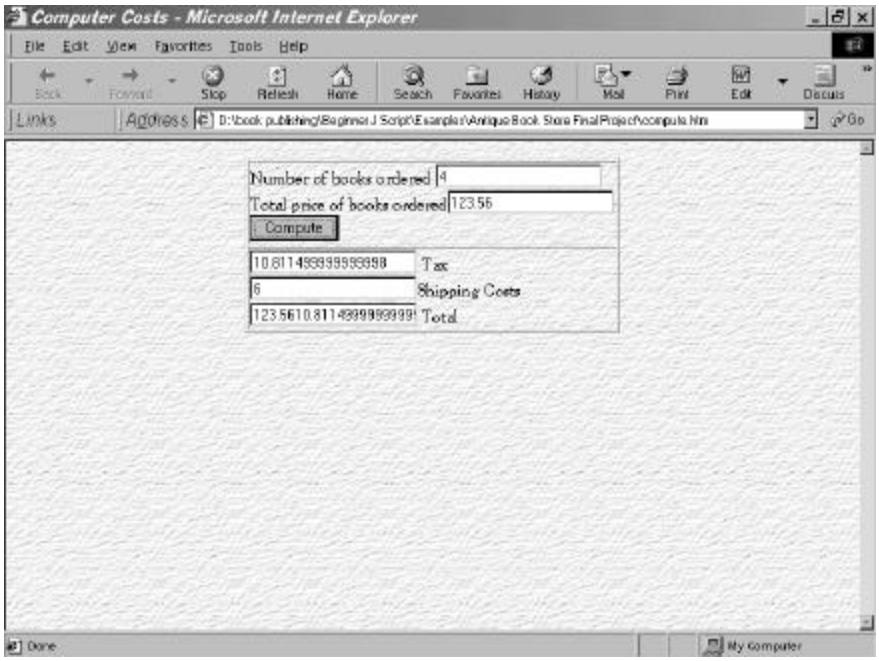


Fig ure 13-2

Summary

This chap ter should have given you an over view of the var i ous math func tions built into JavaScript. It is by no means a com pre hen sive cov er age of math in JavaScript. It should, how ever, give you all the math e mat i cal tools you will need to write some very pow er ful JavaScripts.

Chapter 14

Adding Plug-ins

This chapter ventures into a more advanced HTML topic, that of plug-ins. Plug-ins are essentially objects you insert into your HTML document to provide some added functionality, usually multimedia. If you have ever been to a site that required you to download some component to view the site, then you have used plug-ins.

Strictly speaking, plug-ins are not JavaScript. However, JavaScript is about creating high-quality web sites and I would be remiss in my duties if I did not provide you with this introduction here. Considering the wide range of JavaScript techniques you have already been shown, adding plug-ins will allow you to create some truly spectacular websites.

Putting a Plug-in into Your HTML

A *plug-in* is a piece of software that the browser calls to process data referenced in an HTML document. In order to reference such data in an HTML tag, you must use the <EMBED> tag. This tag's general format is as follows:

```
<EMBED
      SRC=source
      NAME=appletName
      HEIGHT=height
      WIDTH=width>
      [<PARAM NAME=parameterName VALUE=parameterValue>]
      […<PARAM>]
</EMBED>
```

SRC=*source* specifies the URL containing the source content to be interpreted by the plug-in. This may be some external URL, or could simply be a part of your own website where you store the plug-in for downloading.

NAME = *appletName* specifies the name of the embedded object in the document. This name can be just about anything. It is simply the name you will use to reference the plug-in from within your web page.

HEIGHT = *height* specifies the height of the applet in pixels within the browser window.

WIDTH = *width* specifies the width of the applet in pixels within the browser window.

<PARAM> defines a parameter for the embedded object.

NAME = *parameterName* specifies the name of the parameter.

VALUE = *parameterValue* specifies a value for the parameter (an argument).

We will refer to such <EMBED> definitions as plug-ins, although that is not entirely correct.

Using Plug-ins in JavaScript

You can use the plug-ins in your JavaScript code by simply referencing the name you give the plug-in when you embed it. In fact, that is what that name is for. You can also embed plug-ins into an array called embeds, and then reference that element of the embeds array that has your plug-in.

```
document.embeds[0]="myVideo.avi"
```

You can also reference a Plugin object by its name. Take a look at the following HTML definition:

```
<EMBED SRC="mymovie.avi" AUTOSTART=FALSE LOOP=FALSE HEIGHT=120 WIDTH=159
NAME="mymovie">
```

Now you can reference this plug-in's properties and methods by simply calling mymovie.method() (substitute the actual method name). You can also reference it by using the embeds array. For example, if your plug-in were the first one embedded, you could reference it like this:

```
document.embeds[0]
```

Now for a little treat. On the CD in the Chapter14 folder, I have included an ActiveX component (plug-in) that I created. This plug-in allows you to send TCP packets to any valid IP address or URL. I have a little sample code here using it:

Example 14-1

```
<HTML>
<HEAD>
<script language ="JavaScript">
function send()
{
  var ip
  var port
  var data
  var result

  ip =frmip.ipaddress.value
  port =frmport.port.value
  data =frmdata.data.value

  TCPClient.remoteIP =ip
  TCPClient.remotePort =port
  TCPClient.Message =data

  result =TCPClient.sendData()
  alert(result)
}
</script>


<TITLE>Example 14-01</TITLE>
</HEAD>

<BODY BGCOLOR = White>
<OBJECT ID="TCPClient"
  CLASSID="CLSID:26900070-7443-11D5-9A47-00409639327E"
  CODEBASE="TCPClientControl.CAB#version=2,0,0,0">
</OBJECT>
<TABLE BORDER = 1>
  <TR>
  <TD>
  <TABLE border=0 CellPadding = 0 CellSpacing = 0>
    <TR>
         <TD>
              <B>IP Address<B>
         <TD>
         <BR>
         <FORM name ="frmip">
           <INPUT TYPE="text" NAME ="ipaddress" VALUE="" SIZE=10>
         </FORM>
    <TR>
```

```
                <TD>
                     <B>Port<B>
                <TD>
                    <BR>
                    <FORM NAME ="frmport">
                         <INPUT TYPE="text" NAME = "port" VALUE="" SIZE=10>
                    </FORM>
          <TR>
                <TD>
                     <B> Data to send<B>
                <TD>
                <BR>
                <FORM NAME ="frmdata">
                     <INPUT TYPE="text" NAME ="data" VALUE="" SIZE=10>
                </FORM>
          <TR>
                <TD>
                   <CENTER>
                        <INPUT TYPE="button" VALUE="Submit" onClick="send()">
                   </CENTER>
     </TABLE>
</CENTER>
</BODY>
</HTML>
```

If you enter the code prop erly, you will be able to view some thing like the following image:

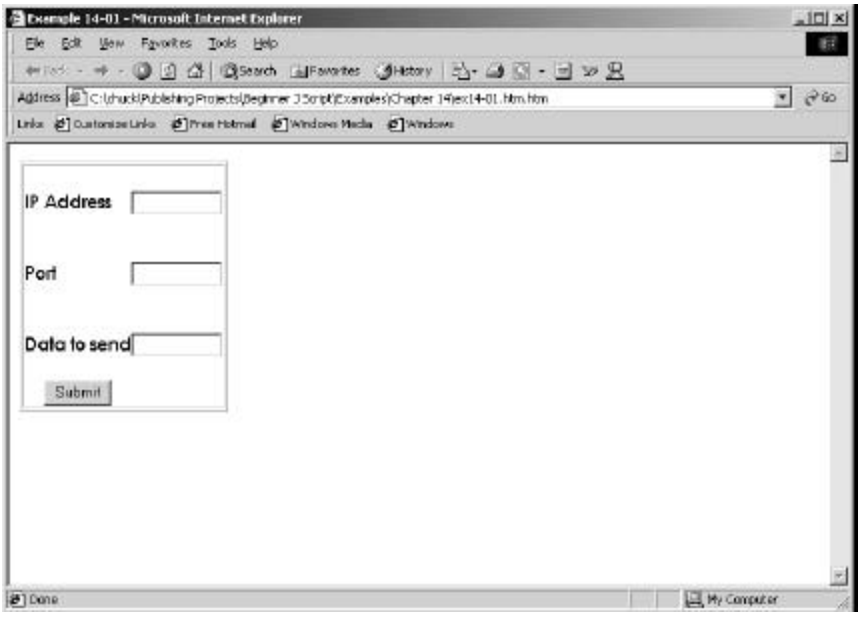


Figure 14-1

This exam ple illus trates using an ActiveX com po nent as a plug-in. The first por tion to look at is in the very begin ning of the body sec tion of the HTML document.

```
<OBJECT ID="TCPClient"
  CLASSID="CLSID:26900070-7443-11D5-9A47-00409639327E"
  CODEBASE="TCPClientControl.CAB#version=2,0,0,0">
</OBJECT>
```


All ActiveX plug-ins have three prop er ties we are con cerned with. The first is the ID. This is sim ply the name you will use to refer to this plug-in. The second is the classid. A classid is a unique iden ti fier that is cre ated when an ActiveX com po nent is first cre ated. The last prop erty we are con cerned with is the codebase prop erty. If a plug-in with the match ing classid can not be found on the cli ent com puter, then the loca tion spec i fied in the codebase prop erty will be accessed to install the ActiveX com po nent.

After the ActiveX plug-in is embed ded in this fash ion we can access any of its prop er ties or meth ods by sim ply call ing the ID we gave the plug-in fol lowed by a dot ".", then the name of the prop erty or method we wish to access. You see this in the script por tion of this exam ple:

```
function send()
{
 var ip
 var port
 var data
 var result

 ip =frmip.ipaddress.value
 port =frmport.port.value
 data =frmdata.data.value

 TCPClient.remoteIP =ip
 TCPClient.remotePort =port
 TCPClient.Message =data

 result =TCPClient.sendData()
 alert(result)
}
```

What Plug-ins are Already Installed?

You can use JavaScript to deter mine if a user has installed a par tic u lar plug-in.You can then dis play embed ded plug-in data if the plug-in is installed, or alter na tive con tent if it is not. The fol low ing exam ple checks for Shock wave and quick time plug-ins.

Example 14-2

```
<HTML>
<HEAD>
   <TITLE> Example 14-02</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
  var quickplug = navigator.plugins["Quicktime"];
  if (quickplug)
  {
      document.writeln("You already have Quicktime")
  }
  else
  {
      document.writeln("You don't have Quicktime<BR>")
  }

  var shockplug = navigator.plugins["Shockwave"];
  if (shockplug)
      document.writeln("You already have Shockwave")
  else
      document.writeln("You don't have Shockwave")
</SCRIPT><BODY BGCOLOR = white>
</BODY>
</HTML>
```

If you enter the code prop erly, you will have the sta tus of these plug-ins displayed on a web page. When run on my PC, the fol low ing out put was generated:

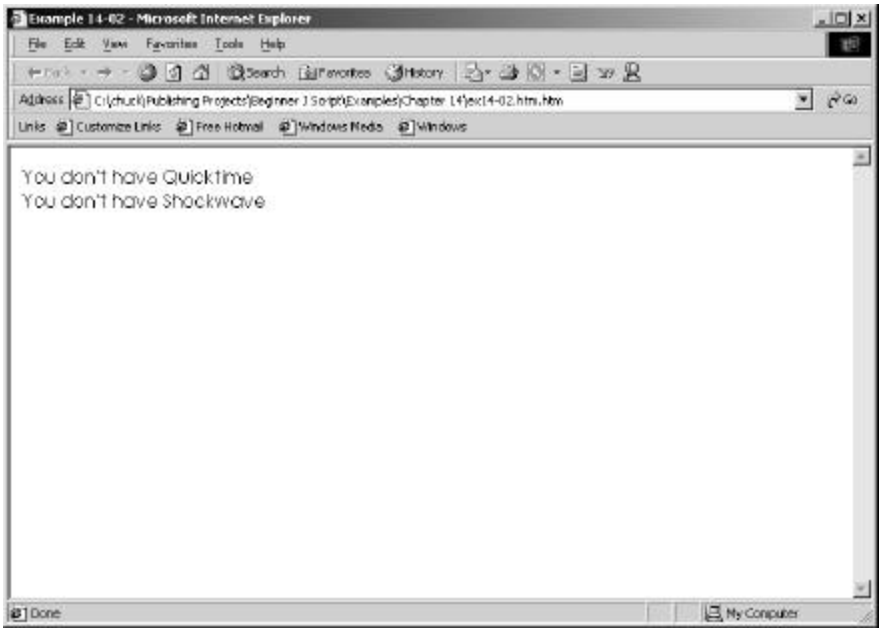


Fig ure 14-2

LiveAudio

LiveAudio is used to play a vari ety of sound files in many for mats includ ing WAV, AIFF, AU, and MIDI for mats. Audio con trols appear accord ing to the size spec i fied in the WIDTH and HEIGHT param e ters in the <EMBED> tag. You can cre ate an audio con sole with any of the fol low ing views:

- con sole—This gives you a ste reo style con sole with sev eral con trols you can use.
- smallConsole—Con sists of a Play, Stop, and vol ume con trol lever. The but tons in this view are smaller than those in a con sole.
- playButton—A but ton that starts the sound play ing.
- pauseButton—A but ton that pauses (with out unload ing) the sound while it is playing.
- stopButton—A but ton that ends the play ing of sound and unloads it.
- volumeLever—A lever that adjusts the vol ume level for play back of the sound (and adjusts the sys tem's vol ume level).

Here is the gen eral HTML syn tax for a LiveAudio con trol:

```
<EMBED SRC=[URL] AUTOSTART=[TRUE|FALSE] LOOP=[TRUE|FALSE|INTEGER]
STARTTIME=[MINUTES:SECONDS] ENDTIME=[MINUTES:SECONDS] VOLUME=[0-100]
WIDTH=[#PIXELS] HEIGHT=[#PIXELS] ALIGN=[TOP|BOTTOM|CENTER|BASELINE
|LEFT|RIGHT|TEXTTOP|MIDDLE|ABSMIDDLE|ABSBOTTOM] CONTROLS=[CONSOLE
|SMALLCONSOLE|PLAYBUTTON|PAUSEBUTTON|STOPBUTTON|VOLUMELEVER] HIDDEN=[TRUE]
```

```
MASTERSOUND NAME=[UNIQUE NAME TO GROUP CONTROLS TOGETHER SO THAT THEY CONTROL
ONE SOUND]...>
```

The syn tax may seem very com pli cated, but a close look shows that it does not con sist of many attrib utes. What's mis lead ing is that there are many dif fer-ent val ues that can be given to each attrib ute. Here is a short descrip tion of each attrib ute and the val ues it accepts:

SRC = [*URL*]—The URL of the source sound file. If the sound file is stored in the same folder as the HTML doc u ment then you can just put the file name with exten sion. AUTOSTART = [TRUE | FALSE]—When set to TRUE, the sound will begin play ing auto mat i cally upon load ing the HTML page. The default is FALSE.

LOOP = [TRUE | FALSE | *INTEGER*]—When set to TRUE, the sound will play con tin u ously until the Stop but ton is clicked on the con sole or the user goes to another page. If an *INTEGER* value is used, the sound repeats the num ber of times indi cated.

STARTTIME = [*MINUTES:SECONDS*]—Use STARTTIME to spec ify where the play back should begin. If you want to begin the sound at 30 sec onds, you would set the value to 00:30 .

ENDTIME = [*MINUTES:SECONDS*]—Use ENDTIME to spec ify where in the sound file you would like play back to end. If you want to stop the sound at 1.5 min utes, you would set the value to 01:30 .

VOLUME = [0-100]—This is a per cent age of max vol ume. 1 to 100 val ues are accepted.

WIDTH = [*#PIXELS*]—Use WIDTH to change the width of the con sole or con sole ele ment. For CONSOLE and SMALLCONSOLE, the default is WIDTH = 144. For VOLUMELEVER, the default is WIDTH = 74. For a but ton, the default is WIDTH = 37 (WIDTH = 34 looks much better).

HEIGHT = [*#PIXELS*]—Use HEIGHT to change the height of the con sole. For CONSOLE, the default is HEIGHT = 60. For SMALLCONSOLE, the default is HEIGHT = 15. For VOLUMELEVER, the default is HEIGHT = 20. For a but ton, the default is HEIGHT = 22.
ALIGN = [TOP | BOTTOM | CENTER | BASELINE | LEFT | RIGHT | TEXTTOP | MIDDLE | ABSMIDDLE | ABSBOTTOM]—While RIGHT and LEFT spec ify the posi tion of the con sole with respect to the page, the other options tell Netscape Nav i ga tor how you want to align text as it flows around the con soles.

It acts sim i larly to the ALIGN attrib ute of the <IMG> tag. The default value is BOTTOM.

CONTROLS=[CONSOLE|SMALLCONSOLE|PLAYBUTTON|PAUSEBUT TON|STOPBUTTON|VOLUMELEVER]—Use this attrib ute to select the con trol you want to place on your page. The default for this field is CONSOLE.

HIDDEN=[TRUE]—The value for this attrib ute should be TRUE, or it should not be included in the <EMBED> tag. If it is spec i fied as TRUE, no con trols will load and the sound will act as a back ground one.

MASTERSOUND—This value must be used when group ing sounds together in a NAME group. It takes no value (it must merely be pres ent in the <EMBED> tag), but tells LiveAudio which file is a gen u ine sound file and allows it to ignore any stub files. In order to asso ci ate sev eral EMBEDs with one sound file, all EMBEDs should have the same name (see the NAME attrib ute). The SRC attrib ute of one of those EMBEDs should be the URL of the actual sound file, whereas the other SRC attrib utes should spec ify the URL of a stub file. A stub file is a text file con tain ing a sin gle space (that's the rec om mended con tent). Its name should con sist of a sound exten sion (e.g., .mid, .wav, .aif). To cre ate a page with four LiveAudio ele ments (Play, Pause, Stop, and Vol ume) all con trol ling the same file, you need to cre ate three sound stubs and of course have one legit i mate sound file (for a total of four EMBEDs). Any time you use the NAME attrib ute in a LiveAudio <EMBED>, you must also use a MASTERSOUND attrib ute. LiveAudio will play no sound when a NAME attrib ute exists with out a cor re spond ing MASTERSOUND attrib ute, even if that is the only <EMBED> with that name on the page. Since you do not want LiveAudio to attempt to play a stub file (it con tains no sound data), you should spec ify a NAME attrib ute with no MASTERSOUND attrib ute. The <EMBED> reflect ing the legit i mate sound file, on the other hand, should fea ture MASTERSOUND in order to play.

NAME=[*UNIQUE NAME*]—This attrib ute sets a unique ID for a group of EMBEDs (each with a dis tinct CONTROLS attrib ute), so they all act on the same sound as it plays. The defi ciency of EMBED's syn tax is that it takes only one value for CONTROLS. For exam ple, if a con tent cre ator wishes to have one sound con trolled by two embed ded objects (a PLAYBUTTON and a STOPBUTTON), he or she must use two sep a rate EMBEDs and group them by the NAME attrib ute. In this case, the MASTERSOUND tag is nec es sary to flag LiveAudio and let it know which of the two <EMBED> tags actu ally has

the sound file you wish to con trol. LiveAudio ignores any EMBED(s) with no MASTERSOUND tag.

If you want one VOLUMELEVER to con trol mul ti ple NAMEs (or the sys tem vol ume), cre ate an EMBED using VOLUMELEVER as the CONTROL. Then set NAME to "_MASTERVOLUME".

The fol low ing exam ple illus trates how to do this. To use this sam ple code, sim ply sub sti tute any valid .mid file where I have writ ten “sam ple.mid.”

Example 14-3

```
<HTML>
<HEAD>
<TITLE>LiveAudio</TITLE>
</HEAD>
<BODY>
  <TABLE BORDER=1><TR>
  <TD BGCOLOR="black" ALIGN="center">
  Sample Music
<EMBED SRC="sample.mid"
       AUTOSTART=FALSE
       LOOP=FALSE
       CONTROLS=PLAYBUTTON
       WIDTH=30
       HEIGHT=20
       MASTERSOUND
       NAME="sample1">
<EMBED SRC="stub1.aif"
       AUTOSTART=FALSE
       LOOP=FALSE
       CONTROLS=STOPBUTTON
       WIDTH=34
       HEIGHT=22
       NAME="90210">
<EMBED SRC="stub2.aif"
       AUTOSTART=FALSE
       LOOP=FALSE
       CONTROLS=PAUSEBUTTON
       WIDTH=34
       HEIGHT=22
       NAME="sample1">
</TABLE>
</BODY>
</HTML>
```

Antique Bookstore Project

I am not going to add a plug-in to our antique book store pro ject because I feel it would sim ply clut ter the web site. How ever, you should feel free to exper i-ment with the tech niques pre sented in this chap ter.

Summary

In this chap ter I showed you the basics of how to add plug-ins to your HTML pages. I think you will find that using plug-ins can add a dimen sion to your web pages that you can not attain oth er wise.

Chapter 15

Objects in JavaScript

Through out this book I have been using the term "object." By now you have prob a bly gleaned what this means from con text. The pur pose of this chap ter is to pro vide a lit tle more depth to your under stand ing of the term.

An *object* is a pro gram ming abstrac tion that groups data with the code that oper ates on it. All pro grams con tain data of dif fer ent types. An object sim ply wraps all that up in a sin gle place, called an object. We did talk about the String object in Chap ter 12 and the Math object in Chap ter 13. Remem ber that each held data, but also had meth ods you could call. A method is just a name for a func tion that is inside an object.

Properties

All the objects you can see and touch have char ac ter is tics. This book, for exam ple, has width, weight, title, num ber of pages, etc. These attrib utes dis-tin guish this book from all other books. These fea tures are called *properties* or *fields* in OO (object-oriented) ter mi nol ogy. An object's prop erty stores data related to the object. Think of prop er ties as adjec tives or nouns that will fur-ther describe the object.

JavaScript sup ports two dif fer ent types of objects:

- *Built-in objects,* such as the String and Math objects
- *User-defined objects*, ones you cre ate

Using properties

An object's prop er ties hold its data. You can refer to prop er ties using the fol-lowing syn tax:

```
object.propertyName
```

object is the name of the object that the property belongs to. For example, you can assign an object to a variable and then refer to its properties using this variable, followed by the property specification. *propertyName* is the name of the property

Another important concept is that a property belongs to an object, and only to one object. Think about it this way. The height of this book is just that, the height of this book, and not of some other book.

A dot separates each object from its property. A hierarchical object structure, with objects and properties at different levels, uses the same syntax:

```
object1.object2Property1.object3Property2.property3
```

The following statements demonstrate referencing to elements of a hierarchical object structure:

```
var d = a.b.d
document.write(d) // prints 16
var e = a.b.e
document.write(e) // prints 42
var f = a.c.f
document.write(f) // prints true
var g = a.c.g
document.write(g) // prints king
var h = a.c.h
document.write(h) // prints 13
var i = a.c.i
document.write(i) // prints 10
```

As you can see, a variable may be named exactly like a property of an object. This is possible because properties are not at the same scope as variables, objects, and properties.

Methods

As you know, objects consist of both data (properties) and functions that handle the data. These functions are called *methods*. Methods enable an object to perform different actions, mostly on its own properties. Think of methods as verbs associated with an object.

JavaScript's implementation of objects is not as powerful as that of Java, so some OO programming advantages that apply to Java do not apply to JavaScript. But remember that you have already used JavaScript objects and

you will prob a bly use more. So you should at least be famil iar with the terminology.

Using Methods

A method is called in the fol low ing fash ion:

```
objectReference.methodName([arguments])
```

objectReference is the name of the object, or any other ref er ence. *methodName* is the name of the method, and *arguments* are the argu ments that the method accepts.

Because a method is a func tion, the name of the method is always fol lowed by a pair of paren the ses. This rule applies also to meth ods that do not accept arguments.

You prob a bly find this syn tax famil iar. I have been using doc u-ment.write([*expression*]) to print HTML expres sions to the page. write() is a method belong ing to the built-in doc u ment object.

When you cre ate an instance of an object using the new oper a tor, you are really declar ing a new data type accord ing to the object's def i ni tion and allo cat-ing the appro pri ate amount of mem ory for that data type. Once you have cre ated an instance of an object, you do not have to use the new key word any-more when refer ring to that instance.

Object Oriented Concepts

Object-oriented pro gram ming is only par tially sup ported in JavaScript. For full sup port of object ori en ta tion you would have to use a pro gram ming lan guage such as C++ or Java. With that said, I thought you should at least be intro-duced to the basic ter mi nol ogy of object ori en ta tion. You have already been intro duced to prop er ties and func tions. These two terms are prob a bly the most impor tant. The next term to remem ber is instantiation.

JavaScript is based on a scaled-down object-oriented par a digm. This par a digm is often called object based, as opposed to object ori ented. In fully object-oriented lan guages you write classes. A *class* is a tem plate you cre ate for cre at ing objects. Classes do not exist in JavaScript (all objects belong to one "class" that is built into JavaScript). Pack ages are source files that have one or more related classes. Since classes do not exist in JavaScript, nei ther do pack ages.

Many of the more advanced fea tures of object ori en ta tion are not sup ported in JavaScript. For instance, you won't find inher i tance (where one class inher its the pub lic meth ods and prop er ties of another). This is pri mar ily because there are no classes.

There are four con cepts that are inte gral to the entire pro cess of object-oriented pro gram ming. They are abstrac tion, encap su la tion, inher i-tance, and poly mor phism.

Abstraction is basi cally the abil ity to think about con cepts in an abstract way. You can cre ate a class for an employee with out hav ing to think about a spe cific employee. It is abstract and can apply to any employee.

Encapsulation is really the heart of object-oriented pro gram ming. This is simply the act of tak ing the data and the func tions that work on that data and putt ing them together in a sin gle class. Think back to our cov er age of strings and the string class. The string class has the data you need to work on (i.e., the par tic u lar string in ques tion) as well as the var i ous func tions you might use on that data, all wrapped into one class.

Inheritance is a pro cess whereby one class inher its, or gets, the pub lic prop erties and meth ods of another class. The clas sic exam ple is to cre ate a class called ani mal. This class has prop er ties such as weight, and meth ods such as move and eat. All ani mals would share these same prop er ties and meth ods. When you wish to cre ate a class for, say a mon key, you then have class mon key inherit from class ani mal, and it will have the same meth ods and prop er ties that ani mal has. This is one way in which object-oriented pro gram ming supports code reuse.

Polymorphism lit er ally means "many forms." When you inherit the prop er ties and meth ods of a class, you need not leave them as you find them. You can alter them in your own class. This will allow you to change the form those meth ods and prop er ties take.

Building Your Own Objects

Now for the big ques tion, what about build ing your own objects? We have already used some of JavaScript's built-in objects, such as the String and Date objects. Now you should be ready to learn some thing about build ing your own new objects and using them.

Constructor Functions

All objects have con struc tor func tions. A con struc tor func tion is sim ply a function that defines the prop er ties and meth ods of the object. It fires auto-mat i cally when an instance of the object is cre ated. You can think of built-in objects as objects whose con struc tor func tions are pre de fined in JavaScript

The key word this is prob a bly the most impor tant word related to objects in object-oriented pro gram ming. It refers to the cur rent object, or instance. If I am sit ting in my den and I say "this chair is uncom fort able," it is clear that I am refer ring to the cur rent room I am in, not some other room. The key word this is much the same. You can cre ate many instances of an object. The this key word allows you to spec ify the cur rent instance the code is run ning in. Inside a con struc tor func tion it refers to the instance for which the func tion was called. Take a look at the fol low ing func tion:


```
function student(name, age, avgGrade)
{
     this.name = name
     this.age = age
     this.grade = avgGrade
}
```

This func tion accepts three argu ments. It defines an object type of a stu dent in a class. The prop er ties are name, age, and grade, and they are ini tial ized by the val ues passed on to the func tion. You can use the fol low ing state ment to cre ate an instance of this object—a stu dent in a class:

```
var student1 = new student("Sharon", 16, 85)
```

Now you can refer to these prop er ties in the fol low ing fash ion:

```
alert(student1.name + " is a cute " + student1.age + " - year old.)
```

It is also pos si ble to add prop er ties to an object once it has been cre ated. Such prop er ties exist only in the spe cific instance to which they are assigned. The fol low ing script seg ment dem on strates this:

```
function student(name, age, avgGrade)
{
     this.name = name
     this.age = age
     this.grade = avgGrade
}
var student1 = new student("Sharon", 16, 85)
student1.sex = "female"
```

```
var message = student1.name + " is a cute " + student1.age
message += " - year old "
message += (student1.sex == "female") ? "girl." : "boy."
alert(message)
```

Now that we have these con cepts down, let me show you them in action.

Based on the exact def i ni tion of the word this, some JavaScript tends to use alter na tive struc tures for con struc tion func tions. Here is the pre ced ing exam-ple in a dif fer ent form:

```
function student(name, age, avgGrade, sex)
{
     obj = this
     obj.name = name
     obj.age = age
     obj.grade = avgGrade
     obj.description = (sex == "female") ? "girl" : "boy"
}
```

Summary

I freely admit that this topic may be quite dif fi cult for begin ning pro gram mers. Some of you may choose to skip it for now. It is included in this book merely to get you started.

In this chap ter I dis cussed the basics of object-oriented ter mi nol ogy and con-cepts. I also dis cussed how those con cepts are imple mented in JavaScript, and which ones are not. You can cer tainly write scripts with out any knowl edge of object ori en ta tion, but hope fully this infor ma tion will give you a deeper under-stand ing of what is occur ring in your scripts.

Chapter 16

Tips and Tricks

This chapter shows you a few interesting tips and tricks. Some of these are JavaScript tricks, and some are HTML tricks. It is quite difficult to completely separate HTML from JavaScript, since JavaScript runs inside of HTML documents. These tips and tricks are placed here because they either did not fit into the topic content of one of the preceding chapters or seem a little bit advanced for this book. In either case I hope that you find these scripts to be useful for you.

Inline Frames

The latest versions of both Internet Explorer and Netscape support a new tag called <IFRAME>. Let's take a look at how this tag works in an actual web page.

Example 16-1

```
<HTML>
<HEAD>
   <TITLE>Example 16-01</TITLE>
</HEAD>
<BODY>
   <IFRAME WIDTH = 400 HEIGHT = 250 SRC = "frame.htm" >
   </IFRAME>
</BODY>
</HTML>
```

This literally displays one HTML document inside another. If you enter all the code properly, or use the sample on the CD, you should be able to view this in your browser and see something like this:

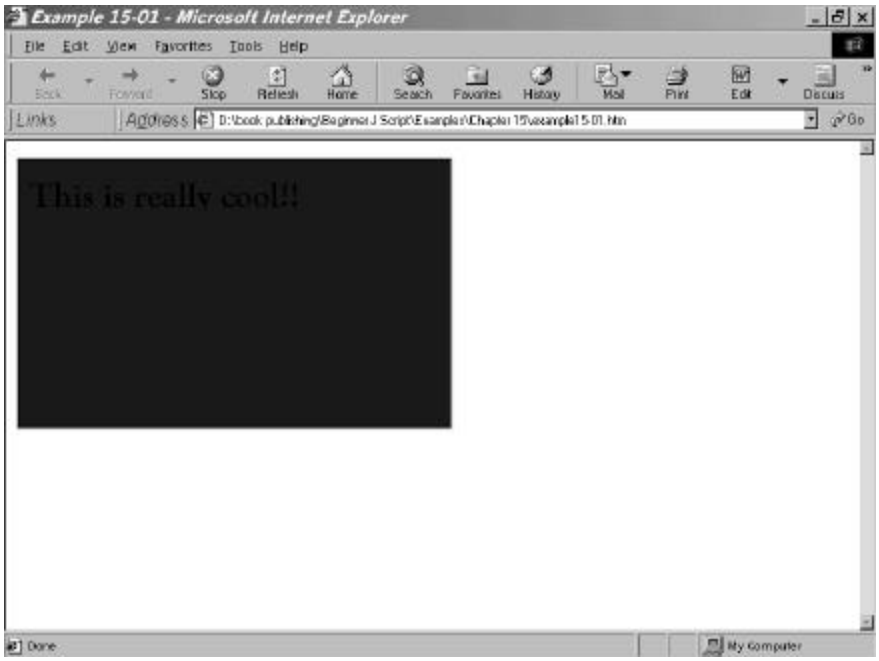


Fig ure 16-1

One of the cool effects you can cre ate with the new <IFRAME> tag is that you can make the inline frame trans par ent. Add ALLOWTRANSPARENCY = "true" to the <IFRAME> ele ment tag. Then add STYLE = "back ground-color:trans par ent" to the <BODY> tag of <file Name> that is being sourced in the <IFRAME> tag (SRC = <file Name>).

The fol low ing code will imple ment this:

```
<IFRAME NAME="Frame1" SRC="somepage.htm"
ALLOWTRANSPARENCY="true" STYLE="position:absolute; top:25; left:50; z-index:3">
```

And the <BODY> tag of somepage.html is:

```
<BODY STYLE="background-color:transparent">
```

The lat est ver sions of both Netscape and Internet Explorer also sup port inline float ing frames. Use the <IFRAME> tag to spec ify the NAME of the frame and its source. Here is the <IFRAME> tag def i ni tion:

```
<IFRAME NAME="Frame1" SRC="somepage.htm">
</IFRAME>
```

Browser Detection

While it is true that browsers are available as free downloads, it is also true that not everyone always downloads the latest to their machine. So it is possible that someone might be using an older browser when viewing your page, and that might mean that their browser does not support all the JavaScript you are using. For that reason, adding some browser detection might be in order. Here is the code:

```
var IE4 = (document.all && !document.getElementById) ? true : false;
var IE5 = (document.all && document.getElementById) ? true : false;
var NET4 = (document.layers) ? true : false;
var NET6 = (document.getElementById && !document.all) ? true : false;
```

Now you can tell which browser is being used by checking each of the variables to see if it is true. Only one will show true, and that will be the browser currently in use.

System Information

JavaScript can retrieve a significant amount of information about the drives in your system, both disk drives as well as shared networks. A drive object is created using the GetDrive() method of the FileSystemObject object. Here is an example:

```
GetDrive(letterDrive);
```

The single parameter, letterDrive, is the given drive name. This method returns the drive object. The following script would create a drive object for c:

```
<SCRIPT LANGUAGE="JavaScript">
var fso = new ActiveXObject("Scripting.FileSystemObject");
driveObj = fso.GetDrive("c");
</SCRIPT>
```

The drive object does not have any methods, but it does have some very useful properties:

Property	Description
AvailableSpace	Returns the number of free bytes on the given drive
DriveLetter	Returns the drive letter of a physical local drive or a shared network
DriveType	Returns the drive type
FileSystem	Returns the file system type for the specified drive

Property	Description
FreeSpace	Returns the number of free bytes on the given drive
IsReady	Returns the status of the drive
Path	Returns the path of the given drive
RootFolder	Returns the folder object of the root folder
SerialNumber	Returns the unique serial number of the volume
ShareName	Returns the shared name of a network drive
TotalSize	Returns the total size, in bytes, of a specified drive
VolumeName	Sets or returns the volume name of the specified drive

If you have expe ri ence with Win dows pro gram ming you might notice that this works a lot like the Win dows FileSystemObject pro vided by scrun.dll (used fre quently by Visual Basic and Visual C + + devel op ers).

Finding Mouse Location

Here is an inter est ing lit tle piece of JavaScript code that will find out where the exact x and y coor di nates of the mouse click event are. For this sam ple I dis play those coor di nates but you could use this code to do par tic u lar actions depend ing on the mouse posi tion.

```
document.onclick = printEvent;
function printEvent(e)
{
   if (navigator.appName == "Netscape")
   {
        mm_X = e.pageX;
        m_Y = e.pageY;

   }
   else
   {
        m_X = event.clientX;
        m_Y = event.clientY;
   }
alert("The Mouse click  was at x  coordinate= " + m_X + " and y
             coordinate = " + m_Y);
}
```

Password

Often you will want some one to enter in a username and pass word, and you will then use your JavaScript to log them into the web site. It is very use ful, to have the pass word masked. On most soft ware, when you enter a pass word, aster isks (*) are what actu ally appear on the screen. For tu nately, this is easy to do by sim ply chang ing one sim ple tag in your HTML form code.

You may have used some thing like this to allow the user to enter a username and pass word:

```
<FORM NAME ="frmname">

     Username <INPUT TYPE="text" NAME = "name" VALUE="" SIZE=10>
     <BR>
     Password <INPUT TYPE="text" NAME ="password" VALUE=""
SIZE=10>

</FORM>
```

Sim ply change the INPUT TYPE from “text” to “pass word” and it will give you the desired effect.

```
<FORM NAME ="frmname">

     Username <INPUT TYPE="text" NAME = "name" VALUE="" SIZE=10>
     <BR>
     Password <INPUT TYPE="password" NAME ="password" VALUE=""
SIZE=10>

</FORM>
```


Browser Information

Some times it is use ful to be able to detect things about a user’s browser. Some of this was cov ered in pre vi ous chap ters. How ever, here is a sim ple script that will give you the user’s screen res o lu tion, browser type, and ver sion num ber.

Example 16-2

```
<html>
<head>
<title> Example 16-02</title>
```

```
<SCRIPT LANGUAGE= "JavaScript">


     var sheight = screen.height
     var swidth = screen.width
     var browsername = navigator.appname
     var version =navigator.appVersion
     alert("Your screen resolution is " + sheight + " X " + swidth)
     alert("You are using " + browsername + " version " + version)

</Script>

</head>

<body bgcolor=White>


</body>
</html>
```

Printing the Page

This lit tle script will allow your users to click a sin gle but ton and print the current page. This can be quite use ful and I rec om mend you use it fre quently.

Example 16-3

```
<html>
<head>
<title>Example 16-03</title>
</head>
<body bgcolor=White>
<input type="submit" name="print" value="Print"onclick="window.print()">
</body>
</html>
```

Antique Bookstore Project

As with most of the chap ters in this book, this chap ter will add to our antique book store pro ject. In this case we are sim ply going to add a Print but ton to one inven tory screen (you could add a but ton to all of them if you so desired). We will add it to the completeinventory.htm page.

```
<HTML>
<HEAD>
```

```
<TITLE>Complete Inventory</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT">

function toggleMenu(currMenu)
{
     if (document.getElementById)
     {
          thisMenu = document.getElementById(currMenu).style
          if (thisMenu.display == "block")
          {
               thisMenu.display = "none"
          }
          else
               {
                    thisMenu.display = "block"
               }
               return false
          }
          else
          {
               return true
          }
}

</SCRIPT>
<STYLE TYPE="TEXT/CSS">
   .menu  {display:none; margin-left:20px}
</STYLE>
</HEAD>
<BODY background="back1.gif">

<H3>
     <A HREF="completeinventory.htm"onClick="return toggleMenu('menu1')">Books
               from the 1800's</A>
</H3>
<SPAN CLASS="menu" ID="menu1">
     1st Edition Charles Dicken's Tale of Two Cities<BR>
     1st Edition Edgar Allen Poe's The Tell Tale Heart<BR>
     Signed Copy of Edgar Allen Poe's The Cask of Amontillado<BR>

</SPAN>
<H3>
     <A HREF="completeinventory.htm" onClick="return toggleMenu('menu2')">Books
              from the 1700's</A>
</H3>
<SPAN CLASS="menu" ID="menu2">
```

```
     Ben Franklin's Memoirs<BR>
     How to start a revolution<BR>
     Political Theory by Thomas Paine
</SPAN>
<H3>
     <A HREF="completeinventory.htm" onClick="return toggleMenu('menu3')">Books
          for under $25</A>
</H3>
<SPAN CLASS="menu" ID="menu3">
     A cowboy's diary (circa 1850's)<BR>
     4th edition Charles Dicken's A tale of two cities
     3rd Edition Jules Verne 20000 Leagues under the Sea
</SPAN>

<H3>
     <A HREF="completeinventory.htm" onClick="return toggleMenu('menu4')">Our
          finest items</A>
</H3>
<SPAN CLASS="menu" ID="menu4">
     Signed copy of Shakespeare's Hamlet<BR>
     Gutenberg's Diary<BR>
     Signed first Edition Longfellow
     1570 Hebrew Bible
</SPAN>

<CENTER><input type="submit" name="print" value="Print"onclick="window.print()">
</CENTER>
</BODY>
</HTML>
```

View the Directory

This next script is rather short but very use ful. It allows you to view the current direc tory in your browser in a Win dows Explorer for mat. You can then browse the entire hard drive via your browser.

Example 16-4

```
<HTML>
<HEAD>
<TITLE>Example 16-04</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">

function getListing()
{
```

```
        var url = location.href
        var lastSlash = url.lastIndexOf("/")
        location.href = url.substring(0, lastSlash + 1)
    }

    </SCRIPT>
    <BODY>
    <FORM>
    <INPUT TYPE="button" VALUE=" view directory listing " onClick="getListing()">
    </FORM>
    </BODY>
    </HTML>
```

If you entered the code prop erly, you should be able to see some thing like this.

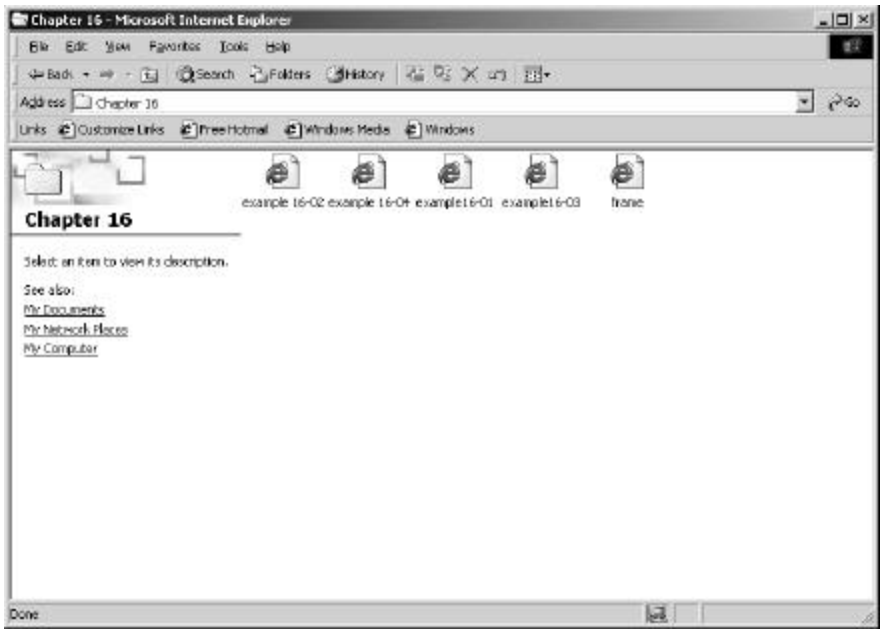


Figure 16-2

Summary

This chap ter sim ply listed a few inter est ing tricks you can do in JavaScript. I strongly sug gest that you visit the websites listed in Appen dix A. These websites have hun dreds of free scripts you can down load, tuto ri als, and much more. While most of this book has focused on teach ing you spe cific tech niques you can use in JavaScript, this chap ter's pur pose has been to intro duce you to a few tech niques that did not seem to fit into the other chap ters.

Chapter 17

JavaScript Games

They say that all work and no fun makes Jack a dull boy. Well, this chap ter is about fun! Here I am going to show you a few rel a tively sim ple games done all in JavaScript. These games will use the tech niques you have already learned, but will use then in novel ways. I think you will find this to be a very excit ing chapter.

Press the Button

Let's start with a rather sim ple exam ple. In this game you sim ply place a button on the screen and ask the user to press it. How ever, when they try to press it, you move it, and change its mes sage.

Example 17-1

```
<html>
<head>
<title>Example 17-01</title>
<script language="JavaScript">
init_msg="PRESS BUTTON NOW"
function new_msg()
{
   var msg=new Array()

   msg[0]="I said press the button"
   msg[1]="Come on, press the button"
   msg[2]="You can't even press a simple button?"
   msg[3]="Use your mouse!"
   msg[4]="Left-click your mouse on the button!"
   var num1=parseInt(Math.random()*(width-150))
   var num2=parseInt(Math.random()*(height-150))
   var num3=parseInt(Math.random()*msg.length)
   ActualObj.left=num1
   ActualObj.top=num2
```

```
   document.clickme.button.value=msg[num3]
}

function start()
{
   if(document.all)
   {
         ActualObj=eval(document.all.floatLyr.style);
         width=document.body.clientWidth
         height=document.body.clientHeight
   }
   else if(document.layers)
   {
         ActualObj=eval(document.floatLyr.document)
         width=innerWidth
         height=innerHeight
   }
   document.clickme.button.value=init_msg
   ActualObj.backgroundColor=document.bgColor
}

function caught()
{
   alert("You caught me!")
}

</script>
</head>
<body onload="start()" bgcolor="white">
<center>Please press the button</center>
<div onmouseover="new_msg()" id="floatLyr"
style="position:absolute;visibility:visible;top:40;left:40">
<form name="clickme">
  <table width="100" border="1" bordercolor=White cellspacing="0" bgcolor=White>

    <tr>
      <td>
         <input name="button" onmouseover="new_msg()" onmousedown="caught()"
type="Button" onkeypress="caught()">

</table>
</form>
</div>
</body>
</html>
```

If you entered the code prop erly you should now be able to see a screen that looks like this:

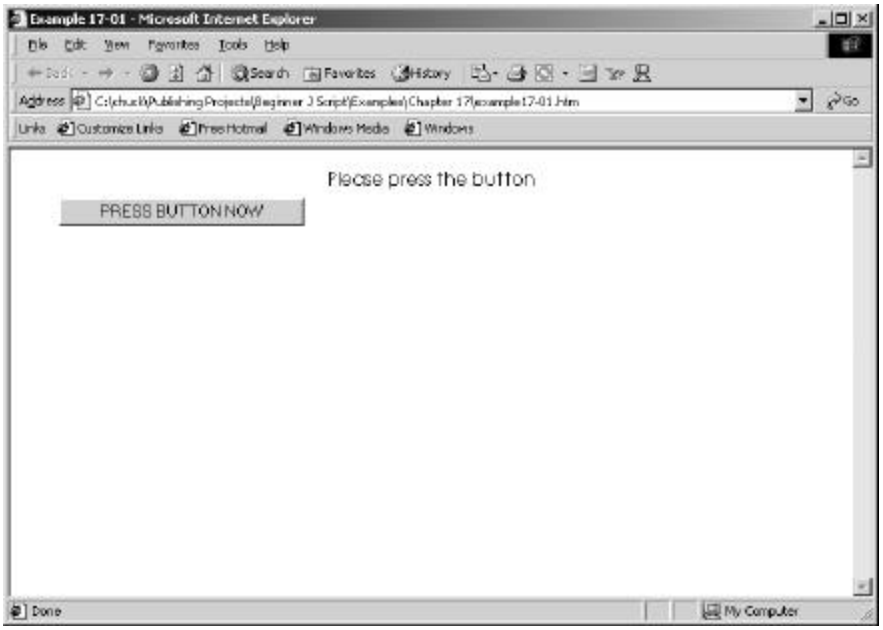


Fig ure 17-1

When you try to move your mouse over the but ton it will move, like this:

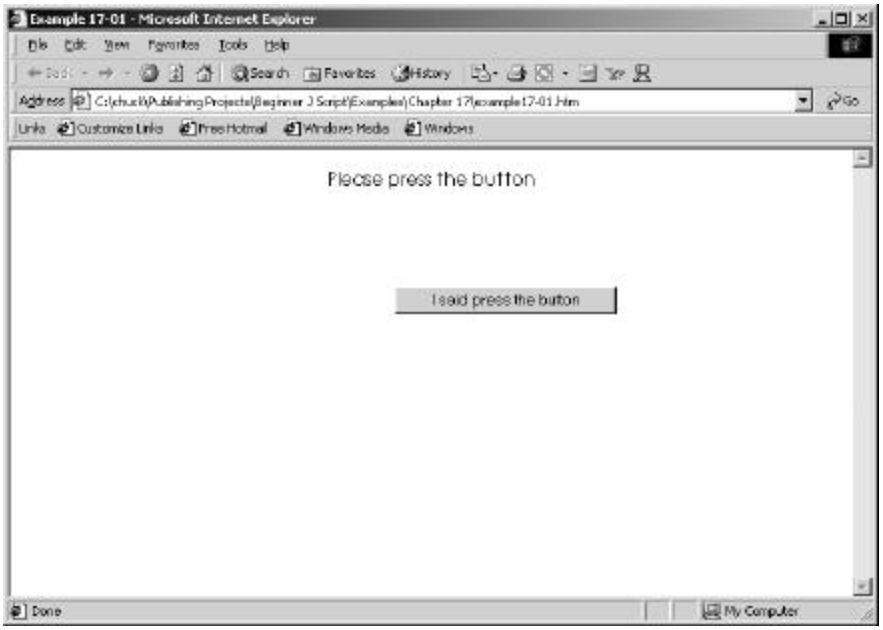


Fig ure 17-2

Roll the Dice

There are many games that require ran dom dice rolls. Most board games uti-lize six-sided dice, whereas many role-playing games uti lize var i ous multisided dice. What ever your needs, this script will gen er ate ran dom dice rolls for you, with any type of dice you wish. You can also eas ily mod ify this to accom mo date more dice or more types of dice.

Example 17-2

```
<HTML>

<HEAD>

<SCRIPT LANGUAGE = "JavaScript">

  var die = 6      // Default dice type
  var dice = 1     // default number of dice

  function rolldice(die, dice)
  {
    var roll = 0
    for (I=0; I< dice; I++)
    {
       roll = roll + Math.round(Math.random() * die) % die + 1;
    }

    document.form.text.value = roll
  }

</SCRIPT>
<BODY>

<center>

  <form name=form>
     <table border=1 bgcolor = gray>
       <tr>
           <td>What kind of dice do you want to roll?
           <td>How many?

      <tr>
        <td>
          <p><input type=radio name=sides onclick="die = 3">3 Sided
          <p><input type=radio name=sides onclick="die = 4">4 Sided
          <p><input type=radio name=sides onclick="die = 5">5 Sided
```

```
            <p><input type=radio checked name=sides onclick="die = 6">6 Sided
            <p><input type=radio name=sides onclick="die = 8">8 Sided

        <td>
            <p><input type=radio name=sides onclick="die = 10">10 Sided
            <p><input type=radio name=sides onclick="die = 12">12 Sided
            <p><input type=radio name=sides onclick="die = 20">20 Sided
            <p><input type=radio name=sides onclick="die = 30">30 Sided
            <p><input type=radio name=sides onclick="die = 100">100 Sided

        <td>
            <p><input type=radio name=number onclick="dice = 1">1
            <p><input type=radio name=number onclick="dice = 2">2
            <p><input type=radio name=number onclick="dice = 3">3
            <p><input type=radio name=number onclick="dice = 4">4
            <p><input type=radio name=number onclick="dice = 5">5

        <td>
            <p><input type=radio name=number onclick="dice = 6">6
            <p><input type=radio name=number onclick="dice = 7">7
            <p><input type=radio name=number onclick="dice = 8">8
            <p><input type=radio name=number onclick="dice = 9">9
            <p><input type=radio name=number onclick="dice = 10">10
    <tr>
       <td>
            <input type=button value="Roll Dice" name=button
                    onclick="rolldice(die, dice)">
            <input type=text size=10 name=text>
    </table>
</form>
</center>
</body>
</HTML>
```

If you entered the code prop erly, you should be able to view some thing like the following figure:

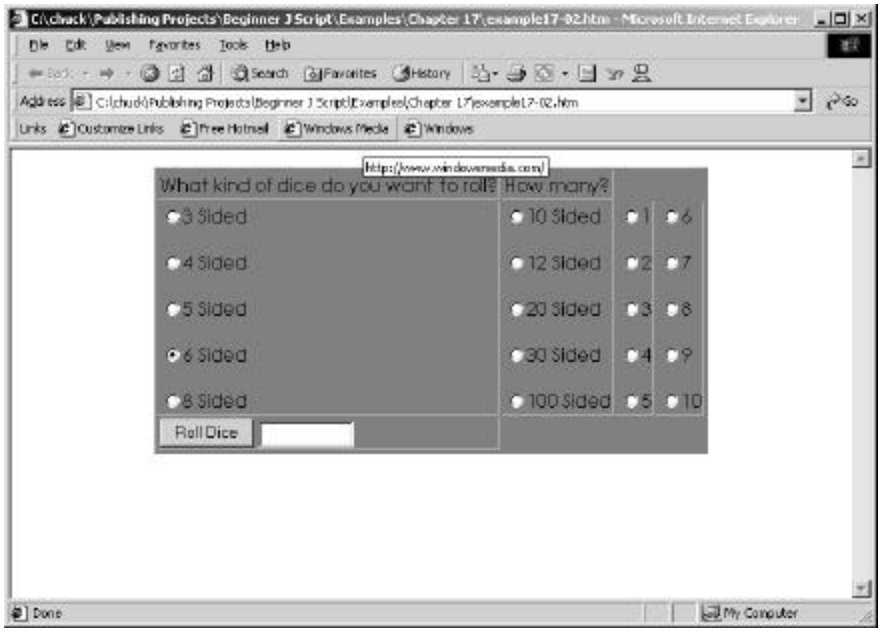


Fig ure 17-3

Viva Las Vegas

This next exam ple is a black jack game that is fairly sim ple. The tech niques are all ones that you have used pre vi ously in this book. They are sim ply com bined in such a way as to give you a work ing black jack game.

Exam ple 17-3

```
<HTML>
<HEAD>
 <TITLE> Example 17-03</TITLE>


<SCRIPT LANGUAGE="JavaScript">

  var gameOver
  var cardCount

  function Shuffle(max)
  {
    var num=Math.random()*max
    return Math.round(num)+1
  }

 function getSuit()
 {
```

```
   suit = Shuffle(4)
  if(suit == 1)
        return "Spades"

  if(suit == 2)
       return "Clubs"

  if(suit == 3)
       return "Diamonds"

  else
      return "Hearts"
}

function cardName(card)
{
  if(card == 1)
    return "Ace"

  if(card == 11)
    return "Jack"

  if(card == 12)
      return "Queen"

  if(card == 13)
      return "King"

  return "" + card
}

function cardValue(card,strWho)
{
 if(card == 1)
 {
      if(strWho =="You" && document.display.you.value >10)
      {
           document.display.say2.value=document.display.say2.value+" Low"
           return 1
       }
       else
            return 11
 }

  if(card > 10)
```

```
       return 10

return card
}

function PickACard(strWho)
{
  card = Shuffle(12)
  suit = getSuit()

  if(strWho =="You")
    document.display.say2.value=(cardName(card) + " of " + suit)

  else
    document.display.say1.value=(cardName(card) + " of " + suit)

  return cardValue(card,strWho)
}

function NewHand(form)
{
if(gameOver !=0)
 {
   form.say1.value=("Hand in Play!")
   form.say2.value=("")
   return
 }
 else
 {
   form.dealer.value = 0
   form.you.value = 0
   cardCount=0
   form.dealer.value = eval(form.dealer.value) + PickACard("Dealer")
   form.you.value = eval(form.you.value) + PickACard("You")
   gameOver = -1
   cardCount+=1
 }
}

function Dealer(form)
{
 if (gameOver ==0)
 {
    form.say1.value=("Please Deal the Cards")
    form.say2.value=("")
    return
```

```
 }


 if (cardCount <2)
  {
    form.say1.value=("There is a Minimum of 2 Cards")
    form.say2.value=("Hit Again")
    return
  }
  else
  {
    while(form.dealer.value < 17)
    {
      form.dealer.value = eval(form.dealer.value) + PickACard("Dealer")
    }
  }

}

function User(form)
{
  if (gameOver ==0)
  {
    form.say1.value=("Please Deal the Cards")
    form.say2.value=(" ")
    return
  }
  else
  {
    cardCount+=1
    form.say1.value="You Get...."
    form.you.value = eval(form.you.value) + PickACard("You")
  }

  if(form.you.value > 21)
  {
    form.say1.value=("Sorry, you lose")
    gameOver=0
    form.numgames.value=(eval(form.numgames.value)) -1
  }
}

function LookAtHands(form)
{
  if (gameOver ==0 || form.you.value<10 || cardCount <2)
  {
```

```
      return
   }

   else
      if(form.dealer.value > 21)
      {
        form.say1.value=("Dealer loses")
        form.say2.value=("You Win")
        gameOver=0
        form.numgames.value=(eval(form.numgames.value))+1
      }

      else
      if(form.you.value > form.dealer.value)
      {
        form.say1.value=("You Win")
        form.say2.value=(" ")

        gameOver=0
        form.numgames.value=eval(form.numgames.value)+1
      }
      else
      {
         if(form.dealer.value == form.you.value)
         {
           form.say1.value=("Game Tied!")
           form.say2.value=("Try Again!")

          gameOver=0
          form.numgames.value=eval(form.numgames.value)-1
         }
         else
         {
           form.say1.value=("House Wins!")
           form.say2.value=("Tough Luck!")
           gameOver=0
           form.numgames.value=eval(form.numgames.value)-1
         }
     }
}
function initialize()
{
  gameOver=0
  cardCount=0;
  document.display.dealer.value=""
  document.display.you.value=""
```

```
  document.display.numgames.value="0"
  document.display.say1.value=" Press the Deal button"
  document.display.say2.value="To Start!"
}

</script>

<BODY OnLoad="initialize()">


<center>
  <form name="display">
        <table bgcolor=GRAY border="1">
          <tr>
            <td><center>Score:</center>
            <td><center>Dealer</center>
            <td><center><input type=text name="dealer" size="2"></center>
            <td><center>Card(s):  <input type=text name="say1"
                        value=""></center>

          <tr>
            <td><center><input type=text name="numgames"value="0"></center>
            <td><center>Player</center>
            <td><center><input type=text name="you"></center>
            <td><center>Card(s):  <input type=text name="say2"value=""></center>

         <tr>
            <td><center><input type=button value="Deal"
onClick="NewHand(this.form)"></center>
            <td><center>
               <input type=button value="Stand"
onClick="Dealer(this.form);LookAtHands(this.form);">
               <input type=button value=" Hit "
onClick="User(this.form)"></center>
       </table>
     </form>
</center>

</body>
</HTML>
```

If you enter all the code prop erly, you should be able to see some thing like the followingfigure.

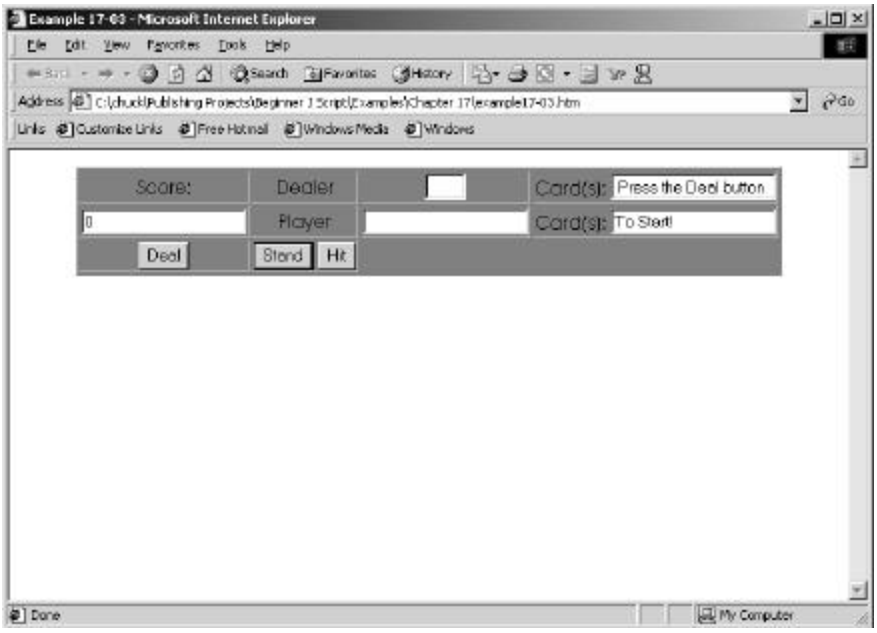


Fig ure 17-4

Summary

This chap ter was placed in this book sim ply to show you some fun things you can do with JavaScript. As you are work ing to learn the tech ni cal nuances of any pro gram ming or script ing lan guage, it is impor tant that you go beyond sim ply mem o riz ing syn tax. With a lit tle bit of imag i na tion, you can take basic tech niques and pro duce truly cre ative results. Albert Ein stein once said that "imag i na tion is greater than knowl edge." Well, most of this book was concerned with increas ing your knowl edge; hope fully, this chap ter sparked your imagination.

Chapter 18

ActiveX and JavaScript

JavaScript is prob a bly the most widely used tech nol ogy for enhanc ing web pages. How ever, there are other pop u lar tech nol o gies being used as well. In an ear lier chap ter I intro duced you to Cas cading Style Sheets. In this chap ter I will intro duce you to ActiveX com po nents. These can only be used in Microsoft Internet Explorer.

Theoretical Background

Before we delve into the use of ActiveX com po nents, let's take a look at some of the the ory under ly ing them. A fre quent prob lem in pro gram ming is allow ing two dif fer ent com po nents to com mu ni cate. One pop u lar tech nol ogy that makes this pos si ble is COM (Com po nent Object Model). With COM, it does not matter what pro gram ming lan guage a com po nent is writ ten in as long as it has an inter face that con forms to COM stan dards. The two most com mon pro gramming tools used to cre ate COM com po nents are Microsoft Visual C++ and Microsoft Visual Basic.

ActiveX is a tech nol ogy based on COM. All ActiveX com po nents adhere to COM inter face spec i fi ca tions. The two most com mon imple men ta tions of ActiveX com po nents are ActiveX con trols and ActiveX DLLs. ActiveX con trols are quite famil iar to Visual Basic pro gram mers. They are also com monly seen on web sites. If you go to a web site that wishes to down load a com po nent to your hard drive, chances are it's an ActiveX con trol. These are com monly used with play ing mul ti me dia. An ActiveX con trol usu ally has a vis i ble inter face.

ActiveX DLL's expose a num ber of func tions that the host appli ca tions (or web page) can use. How ever, ActiveX DLL's rarely have a user inter face. ActiveX con trols are far more com mon in web pages than ActiveX con trols.

Using ActiveX for TCP/IP

Let's look at an actual exam ple of using an ActiveX com po nent in a web page with Java Script. In the Chap ter 18 folder on the com pan ion CD you will find an ActiveX con trol named TCPClientControl.ocx. This is a con trol I devel oped to allow easy access to TCP/IP com mu ni ca tion inside of web pages. It is very sim ple to use, and you should feel free to use this con trol as you wish.

The first step in using an ActiveX con trol is sim ply insert ing it in the web page. This was cov ered briefly in the Chap ter 14, "Adding Plug-ins." The code to insert an ActiveX com po nent into a web page is really rather sim ple:

```
<OBJECT ID="TCPClient"
    CLASSID="CLSID:26900070-7443-11D5-9A47-00409639327E"
    CODEBASE="TCPClientControl.CAB#version=2,0,0,0">
</OBJECT>
```

The <OBJECT> tag is an HTML tag iden ti fy ing that an ActiveX com po nent is being inserted. OBJECT ID is sim ply the name you wish to use for this com po nent within your code. You can choose any name you wish. Once you have cho sen a name you can use that name to access the meth ods and prop erties of the ActiveX com po nent. CLASSID is a unique num ber that each ActiveX com po nent has. It is gen er ated when the ActiveX com po nent is created. CODEBASE iden ti fies the URL from which the ActiveX com po nent can be down loaded if it is not already installed on the user's com puter. This is an espe cially nice fea ture of ActiveX com po nents: they are self-installing.

The method I have just explained to you is the same for all ActiveX com ponents. Now I will show you an actual exam ple of an ActiveX con trol in use. This exam ple cre ates an ActiveX TCP/IP cli ent that can send data to any active TCP/IP Server on the internet.

Example 18-1

```
<HTML>
<HEAD>
   <TITLE>Example 18-01</TITLE>
<SCRIPT LANGUAGE = "JavaScript">
function send()
{
   var ip
   var port
   var message

   ip = frmdestination.remoteip.value
```

```
   port = frmdestination.remoteport.value
   message = frmdestination.message.value

   TCPClient.remoteIP = ip
   TCPClient.remotePort =port
   TCPClient.Message =message
   TCPClient.sendData()
}
</SCRIPT>
</HEAD>
<BODY>
<P>
<CENTER>
    <TABLE BORDER = 1 BGCOLOR = lightblue>
       <TR>
         <TD>
             <FORM name ="frmdestination">
                 IP Address<INPUT TYPE="text" NAME ="remoteip" VALUE=""
                        SIZE=15><BR>
                 Port Number<INPUT TYPE="text" NAME ="remoteport" VALUE=""
                        SIZE=15><BR>
                 Message<INPUT TYPE="text" NAME ="message" VALUE="" SIZE=25><BR>
                 <INPUT TYPE ="button" VALUE ="Connect" onClick="send()">
              </FORM>
    </TABLE>
</CENTER>
<OBJECT ID="TCPClient"
    CLASSID="CLSID:26900070-7443-11D5-9A47-00409639327E"
    CODEBASE="TCPClientControl.CAB#version=2,0,0,0">
</OBJECT>
</BODY>
</HTML>
```

If you entered all the code properly you will be able to see the following:

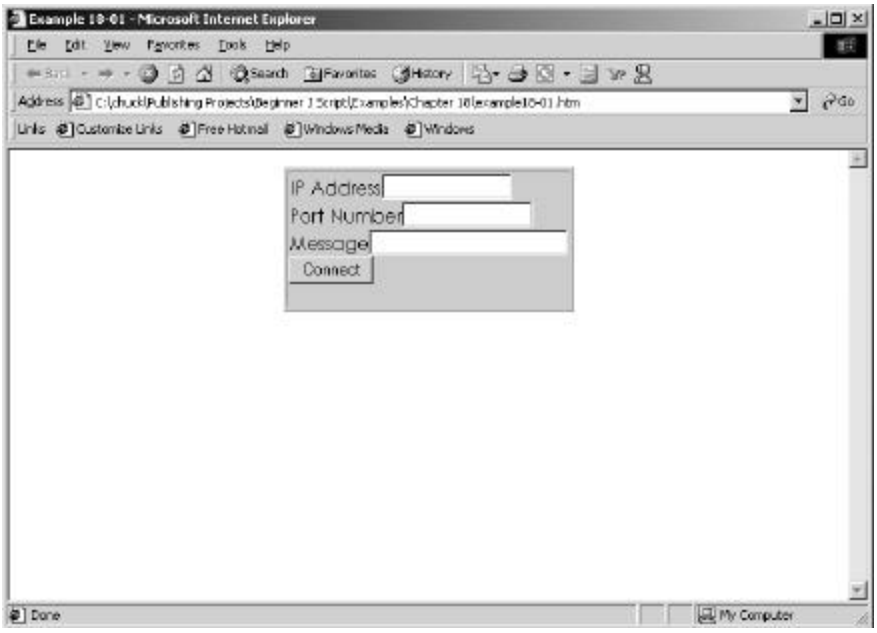


Figure 18-1

This example shows you an easy way to utilize ActiveX controls in your web site. In this case the ActiveX control takes care of the TCP/IP communication with some TCP/IP server.

Ticking Clock

This next little component offers a nice visual effect. It places a clock anywhere you wish on your web page. This is a rather easy method for inserting a clock on your web page.

Example 18-2

```
<HTML>
<HEAD>
   <TITLE>Example 18-02</TITLE>
<SCRIPT LANGUAGE = "JavaScript">
</SCRIPT>
</HEAD>
<BODY>


<CENTER>
  <OBJECT ID="Active_Clock"
    CLASSID="CLSID:A119E782-3E4F-4EF8-A959-FA3FD028076E"
```

```
    CODEBASE="ActiveClock.CAB#version=1,0,0,0">
  </OBJECT>
</center>
</BODY>
</HTML>
```

If you enter the code prop erly, you should be able to view this web page in Internet Explorer and see the fol low ing image.

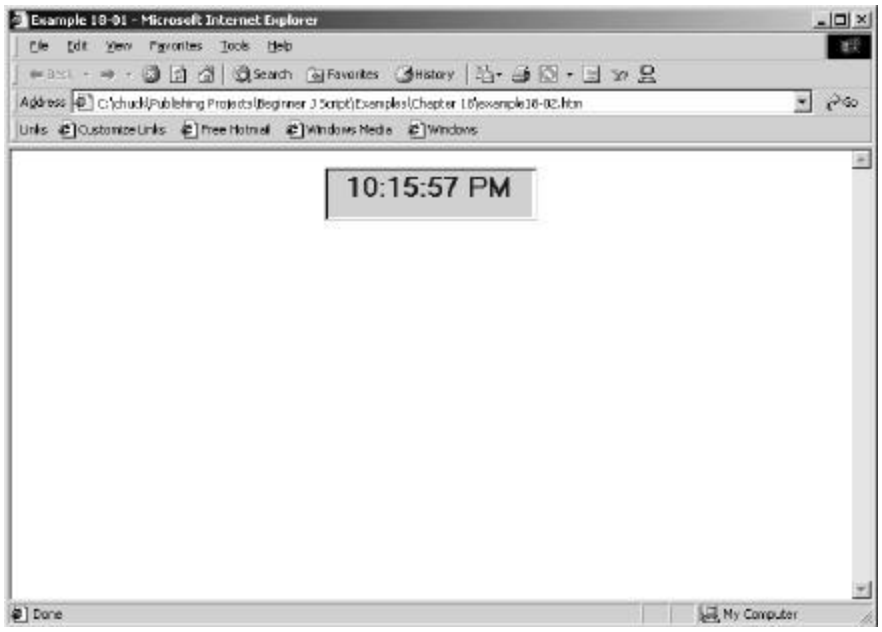


Fig ure 18-2

Slider Text

This ActiveX com po nent is inter est ing for sev eral rea sons. First of all, I am inter ested in it because I use it fre quently to teach stu dents how to cre ate their own ActiveX com po nents. It is also inter est ing for us because with this ActiveX com po nent, I can show you how to manip u late an ActiveX com po-nent's prop er ties via JavaScript. Most ActiveX com po nents come with some doc u men ta tion tell ing you what prop er ties and meth ods they have and what they do.

Example 18-3

```
<HTML>
<HEAD>
   <TITLE>Example 18-03</TITLE>
```

```
<SCRIPT LANGUAGE = "JavaScript">
  function setslider()
  {
    SliderText.Max = 50
    SliderText.flagextreme =true
  }
</SCRIPT>
</HEAD>
<BODY onLoad = "setslider()">
<CENTER>
<OBJECT ID="SliderText"
  CLASSID="CLSID:1E0822BF-195D-11D5-9A46-00409639327E"
  CODEBASE="slidertext1.CAB#version=1,0,0,0">
</OBJECT>



</center>

</BODY>
</HTML>
```

If you entered all the code prop erly, you will be able to see this image:

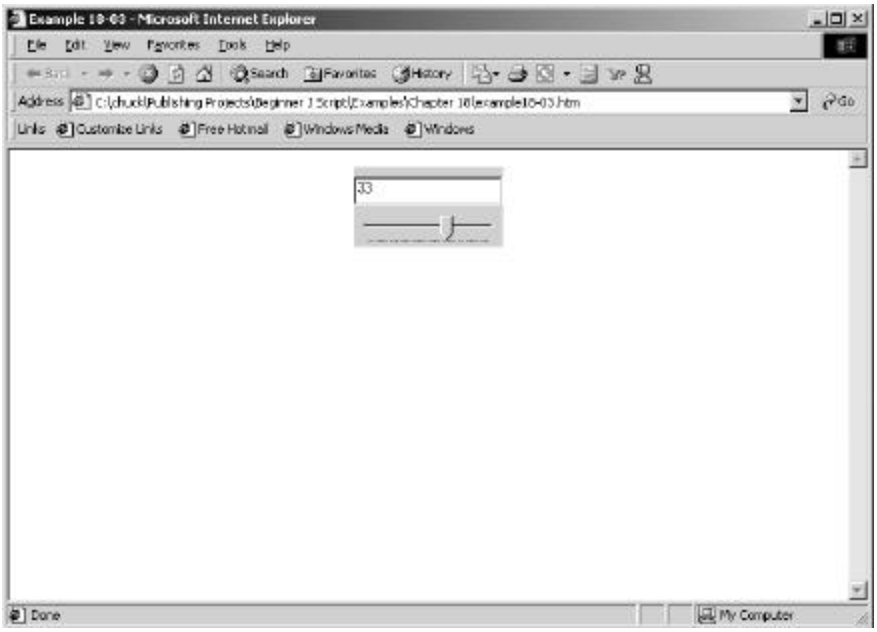


Fig ure 18-3

Summary

This chap ter is not meant to be an exhaus tive treat ment of the topic of ActiveX. How ever, I hope that after read ing this chap ter you will real ize that ActiveX com po nents are one more tool you have at your dis posal for cre at ing truly stun ning web sites. Often ActiveX is used in con junc tion with VB Script; how ever, now you should real ize that it is really not very hard to use it with JavaScript as well. I also hope you real ize that tak ing advan tage of exist ing ActiveX com po nents can allow you to greatly expand your web sites.

Chapter 19

Programming and JavaScript

This book has focused on accom plish ing spe cific goals using JavaScript. This chap ter is included to intro duce you to some con cepts that are com mon to pro-gram ming. Cer tainly a per son sim ply using JavaScript for their own per sonal web page should feel free to skip this chap ter. How ever, a pro fes sional web devel oper needs to under stand what pro gram ming is, how it's done, and how it relates to JavaScript. This chap ter also lays the ground work for those of you who wish to con tinue on to my *Advanced JavaScript, Sec ond Edi tion* book (also avail able from Wordware pub lish ing).

General Concepts

We have used vari ables through out this book, but what exactly are they? A *variable* is a place in mem ory set aside to hold data. The vari able's name allows us to refer to that place in mem ory and to manip u late the data held therein. With out vari ables you would have to refer to the actual hex a dec i mal address in mem ory where data is stored. In many lan guages, data is stored in vari ables of a spe cific type. Whole num bers are stored in inte ger data types, dec i mals are stored in float data types, etc. In JavaScript the var key word allows you to declare a vari able that will hold any type of data. This is very con ve nient for you when you are writ ing scripts.

A *statement* is a line of code that per forms some action. All of the fol low ing are statements:

```
Var mydate   // the action performed is the declaration of a variable
Var mynumber = x * 3  // The action performed is a mathematical operation.
```

When you group state ments together into a log i cal group ing under some name, you have a *function*. A func tion may or may not take any param e ters, and may or may not return some value. In other pro gram ming lan guages you have to define the func tion in such a way as to have it return data of a spe cific

type. In JavaScript you have the free dom to return any type of data you wish, or to return no data at all.

Now that we have defined vari ables, state ments, and func tions, we have defined the foun da tions of all pro gram ming. Com bining this with a knowl edge of oper a tors (intro duced to you in Chap ter 13, "Math in JavaScript") and you have the basic build ing blocks of pro gram ming. The pri mary dif fer ence between script ing lan guages, like JavaScript, and tra di tional pro gram ming languages, such as C and Java, is that script ing lan guages are much more flex i ble. You can do things any way you please. How ever, this can fre quently lead to code that is very dif fi cult to read and main tain.

Good cod ing prac tices are really quite sim ple. The first thing to keep in mind is to make your code read able. You will notice that I space my code out. It is quite pos si ble to place more than one state ment on a sin gle line, but I avoid that because it does not facil i tate read able code. The next thing I highly recom mend is the lib eral use of com ments. Com ments can make your code more under stand able. Sim ply place a // and any thing writ ten after it will be ignored by the browser. This way you can leave explan a tory com ments tell ing the reader what you are doing in your code.

Arrays

Arrays are data struc tures, and are some what more com plex than sim ple variables. When you have related data that you wish to group together, an array pro vides a very good way to do that. An exam ple would be a set of stu dent grades. If you want to pro cess an indi vid ual item of an array you need to specify the array name and indi cate which array ele ment is being ref er enced. Spe cific ele ments are indi cated by an index or a sub script.

Arrays in JavaScript are sim ple built-in objects. You cre ate an array just like an instance of an object, because that is exactly what it is. The for mal name of the object is Array—notice the cap i tal ized "A." The gen eral syn tax is:

```
var arrayObjectName = new Array()
var arrayObjectName = new Array(arrayLength)
```

arrayObjectName is the name of a new object, an exist ing vari able, or a property of an exist ing object.

ArrayLength is the num ber of indi vid ual ele ments in the array.

In JavaScript you do not have to spec ify the actual size of the array, nor the type of vari ables it will con tain. This is because JavaScript will allow you to expand the array if needed, and you can place in any type of data you wish.

Here are some arrays:

```
var day = new Array(31)
var month = new Array(12)
var year = new Array() // number of years "not known"
```

All ele ments of an array are ini tially null. This is impor tant because such elements do not have any influ ence on your script. An ele ment with a null value actu ally does not exist. This is why it is com mon prac tice to give each ele ment of an array an ini tial value. As you might guess, this is referred to as *initializing* an array.

From the moment an array is declared, it takes up the required stor age area in memory.

It also does not mat ter if you ini tial ized its val ues or not. The o retically, if you cre ated an array with out spec i fy ing the num ber of ele ments, it would be as if you cre ated one with zero ele ments.

To use an array you must be able to refer to its ele ments. Arrays in JavaScript are objects. Like all other objects, they have prop er ties and meth ods:

```
arrayObjectName[subscript] // ar1[4]
```

The *subscript* fol lows the array name and is enclosed in square brack ets. Subscripts are sim ple inte gers that start at zero.

Here is a sim ple array:

```
var ar = new Array(5)
```

This array has five ele ments: ar[0], ar[1], ar[2], ar[3], ar[4].

After you cre ate an array you can increase its length by spec i fy ing a value for the high est sub script ele ment. The fol low ing code cre ates an array of length zero, then assigns a null value to ele ment 99. This changes the length of the array to 100.

```
accounts = new Array()  // array of zero elements
accounts[99] = null     // array of 100 elements
```

Note that the array does not take up any stor age space, even after it is extended.

When refer ring to an ele ment, the sub script can be either a lit eral (e.g., 3) or a vari able (e.g., num = 3).

An ele ment of an array can be any valid value. It can be a string, a num ber, a Boolean value, a null value, or even another object. For exam ple, if you want to cre ate an array in which each ele ment is a stu dent object, you can use the following state ments:

```
function student()
{  // constructor function
   // properties not initialized to meaningful value
   this.name = ""
   this.age = ""
   this.grade = ""
}
var size = 35                       // num of students in class
var students = new Array(size)      // array is defined
for (var i = 0; i < size; i++)
{
   students[i] = new student()
}
students[0].name = "Mark"
students[32].grade = 88
```

At first, the desired size of the array—the num ber of stu dents in the class—is assigned to the vari able size. An array of that size is then cre ated. All elements of the array, from stu dents[0] to stu dents[34], are then defined using the con struc tor func tion stu dent(). In this exam ple, all of the ele ments in the array are of the same type. An array can also have ele ments of dif fer ent types. Here is an exam ple:

```
function student()
{  // constructor function
   // properties not initialized to meaningful value
   this.name = ""
   this.age = ""
   this.grade = ""
}
function teacher(name, age)
{
   this.name = name
   this.age = age
}
var size = 35                       // num of students in class
var students = new Array(size + 1) // array is defined
students[0] = new teacher("Kate", 45)
```

```
for (var i = 1; i < size + 1; i++)
{ // or i <= size
   students[i] = new student()
}

alert("   is the teacher." + students[0].name)
```

In this script seg ment an array of size + 1 ele ments is defined, because the first ele ment, stu dents[0], holds an instance of the teacher object.

The most impor tant rule is that the sub script, or index, starts at zero. Although it might seem quite awk ward, use this ele ment like all other elements of the array.

Summary

The pur pose of this chap ter was to intro duce you to con cepts com mon to all types of pro gram ming. This knowl edge is not nec es sary for the hob by ist but is vital for the pro fes sional pro gram mer (or the would-be pro fes sional). If you wish to explore these con cepts in more depth, as well as explor ing JavaScript in greater depth, may I hum bly rec om mend my own *Advanced JavaScript, Second Edi tion* from Wordware Pub lishing.

Chapter 20

Antique Bookstore Project

Through out most of this book we have grad u ally been build ing a real-world web site for a fic ti tious antique book store. This chap ter sim ply brings all the source code into one con ve nient loca tion. You can also find that same code on the CD in the Antique Book store Pro ject folder.

The Main Screen

First I will show you the main screen, which con tains index.htm, main.htm, ban ner.htm, and tool.htm. Main.htm has the frame code to hold the other pages.

```
<html>
<head>
<title>Ye Olde Book Shop</title>
</head>
<frameset rows="64,*">
  <frame name="banner" scrolling="no" noresize target="contents"
             src="banner.htm">
  <frameset cols="169,*">
    <frame name="contents" target="main" src="tool.htm" scrolling="auto">
    <frame name="display" src="main.htm" scrolling="auto" target="_self"
             marginwidth="0" marginheight="16">
  </frameset>
  <noframes>
  <body>

  <p>This page uses frames, but your browser doesn't support them.</p>

  </body>
  </noframes>
</frameset>
</html>
```

<index.htm>

```
<HTML>
<HEAD>
   <TITLE>Ye Olde Book Shoppe</TITLE>
<SCRIPT LANGUAGE="JavaScript">

function setCookie(name, value, expires, path, domain, secure)
{
   var curCookie = name + "=" + escape(value) +
        ((expires) ? "; expires=" + expires.toGMTString() : "") +
   ((path) ? "; path=" + path : "") +
   ((domain) ? "; domain=" + domain : "") +
   ((secure) ? "; secure" : "")
   document.cookie = curCookie
}


function getCookie(name)
{
var prefix = name + "="
   var cookieStartIndex = document.cookie.indexOf(prefix)

           if (cookieStartIndex == -1)
           return null

           var cookieEndIndex = document.cookie.indexOf(";", cookieStartIndex+
                    prefix.length)

           if (cookieEndIndex == -1)
           cookieEndIndex = document.cookie.length

           return unescape(document.cookie.substring(cookieStartIndex
                    +prefix.length, cookieEndIndex))
}

function deleteCookie(name, path, domain)
{
   if (getCookie(name))
         {
         document.cookie = name + "=" +((path) ? "; path=" + path : "") +
                   ((domain) ? "; domain=" + domain : "") +"; expires=Thu,
                   01-Jan-70 00:00:01 GMT"
         }
}
   var expiredate= new Date()
   expiredate.setTime(expiredate.getTime() + 30 * 24 * 60 * 60 * 1000)
   var name = getCookie("name")
```

```
   if (!name)
   {
         name = prompt("Please enter your name:", "John Doe")
         setCookie("name", name, expiredate)
   }
alert("Welcome back " + name)
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
var mydate = new Date()
var myday = mydate.getDay()
if(myday==0)
   alert("Sunday - Sorry, we are closed today")
if (myday==1)
   alert("Monday - Sorry, we are closed today")

if(myday==2)
   alert("Tuesday - Our hours today are 10 a.m. to 6 p.m.")

if(myday==3)
   alert("Wednesday - Our hours today are 10 a.m. to 6 p.m.")

if (myday==4)
   alert("Thursday - Our hours today are 10 a.m. to 6 p.m.")

if (myday==5)
   alert("Friday - Our hours today are 10 a.m. to 7 p.m.")

if (myday==6)
   alert("Saturday - Our hours today are 10 a.m. to 4 p.m.")

</SCRIPT>
   <SCRIPT LANGUAGE="JAVASCRIPT">
         ImageArray = new Array("banner1.gif","banner2.gif","banner3.gif")
         CurrentImage = 0
         ImageCount = ImageArray.length
         function RotateBanner()
         {
         if (document.images)
         {
               CurrentImage++
               if (CurrentImage ==ImageCount)
               {
                     CurrentImage = 0
               }
               document.Banner.src=ImageArray[CurrentImage]
                           setTimeout("RotateBanner()",3000)
```

```
            }
        }


</SCRIPT>
</HEAD>
<BODY  background="back1.gif"  onLoad="RotateBanner()">
<SCRIPT LANGUAGE="JavaScript">
   var bgOffset = 0;
   var bgObject = eval('document.body');

   function scrollbackground(maxSize)
   {
         bgOffset = bgOffset + 1;

         if (bgOffset > maxSize)
               bgOffset = 0;

         bgObject.style.backgroundPosition = "0 " + bgOffset;
   }
var scrtimer = window.setInterval("scrollbackground(50)",50);

</SCRIPT>
<CENTER>
<IMG SRC="banner1.gif"  NAME="Banner" >
<P><TABLE BORDER=1>
   <TR>
      <TD>
         <P><IMG SRC="gargshelf.gif">
     <TD>
         <P><CENTER>Ye Olde Book Shoppe</CENTER>
     <TD>
         <IMG SRC="gargshelf.gif">

</TABLE>

<P>
<P>


</CENTER>
<P>
</BODY>
</HTML>
            <main.htm>
<HTML>
```

```
<HEAD>
   <TITLE>Ye Olde Book Shoppe</TITLE>
</HEAD>
<BODY background="back1.gif">
<P>
    <A HREF = "books.htm" target = "display">See our books</A><BR>
    <A HREF = "inventory.htm" target = "display">Latest Additions</A><BR>
    <A HREF = "completeinventory.htm" target = "display">Complete
                Inventory</A><BR>
    <A HREF = "compute.htm" target = "display">Compute Costs</A><BR>
</BODY>
</HTML>
             <tool.htm>
<HTML>
<HEAD>
   <TITLE>Ye Olde Book Shoppe</TITLE>
</HEAD>
<BODY background="sample1.gif">
<CENTER>
<H1><B><I>Ye Olde Booke Shoppe</I> </B></H1>
</CENTER>
</BODY>
</HTML>
             <banner.htm>
```


These pages you have just seen make up the pri mary por tions of the web site. This is what the user will see when they first visit the web site. Now let's look at the inven tory pages.

Inventory Pages

```
<HTML>
<HEAD>
   <TITLE>Book Inventory</TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT" >

   var childwindow = null

   function openbook1()
   {
        childwindow = window.open("inventory1.htm","childwindow",
                  "width=400,height=400,scrollbars=yes")
   }
   function openbook2()
```

```
   {
        childwindow = window.open("inventory2.htm","childwindow",
                  "width=400,height=400,scrollbars=yes")
   }
   function openbook3()
   {
        childwindow = window.open("inventory3.htm","childwindow",
                  "width=400,height=400,scrollbars=yes")
   }
   function openbook4()
   {
        childwindow = window.open("inventory4.htm","childwindow",
                  "width=400,height=400,scrollbars=yes")
   }
</SCRIPT
</HEAD>
<BODY   background="back1.gif" >
<P>
<CENTER>
        <H2> This month's newest additions!
</CENTER>
        <P>
        <A HREF="javascript:openbook1()"
        onMouseOver= " window.status='We have three copies of this book, all
                  in fine condition'
        return true" onMouseOut="window.status= ' '
        return true">View Shakespeare</A><BR>

        <A HREF="javascript:openbook2()"
        onMouseOver= " window.status='We just acquired this book last month'
        return true" onMouseOut="window.status= ' '
        return true">View Dickens</A><BR>

        <A HREF="javascript:openbook3()"
        onMouseOver= " window.status='This book is in fair condition'
        return true" onMouseOut="window.status= ' '
        return true">View Poe</A><BR>

        <A HREF="javascript:openbook4()"
        onMouseOver= " window.status='We have 1 copy in mint condition'
        return true" onMouseOut="window.status= ' '
        return true">View King James Bible</A><BR>
        </H2>

</BODY>
</HTML>
```

```
            <inventory.htm>

<HTML>
<HEAD>
    <TITLE>Advanced JavaScript</TITLE>
</HEAD>
<BODY BACKGROUND ="bod-bg.gif">
<BR>
<BR>
<CENTER>
   <IMG SRC = "book1.gif" Height = 100 Width = 100>
        <BR>
        <BR>
        Signed copy of McBeth
        $2900
</CENTER>
</BODY>
</HTML>
            <inventory1.htm>

<HTML>
<HEAD>
    <TITLE>Advanced JavaScript</TITLE>
</HEAD>
<BODY BACKGROUND ="bod-bg.gif">
<BR>
<BR>
<CENTER>
   <IMG SRC = "book2.gif" Height = 100 Width = 100>
        <BR>
        <BR>
        Mint condition 1st edition Tale of Two Cities
        $1200
</CENTER>
</BODY>
</HTML>
            <inventory2.htm>

<HTML>
<HEAD>
    <TITLE>Advanced JavaScript</TITLE>
</HEAD>
<BODY BACKGROUND ="bod-bg.gif">
<BR>
<BR>
<CENTER>
```

```
   <IMG SRC = "book3.gif" Height = 100 Width = 100>
         <BR>
         <BR>
         Signed First Edition of "The Raven"
         $1450
</CENTER>
</BODY>
</HTML>
              <inventory3.htm>

<HTML>
<HEAD>
    <TITLE>Advanced JavaScript</TITLE>
</HEAD>
<BODY BACKGROUND ="bod-bg.gif">
<BR>
<BR>
<CENTER>
   <IMG SRC = "book4.gif" Height = 100 Width = 100>
         <BR>
         <BR>
         The original King James Bible
         $10450
</CENTER>
</BODY>
</HTML>
              <inventory4.htm>

<HTML>
<HEAD>
<TITLE>Complete Inventory</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT">

function toggleMenu(currMenu)
{
   if (document.getElementById)
   {
         thisMenu = document.getElementById(currMenu).style
         if (thisMenu.display == "block")
         {
               thisMenu.display = "none"
         }
         else
         {
               thisMenu.display = "block"
         }
```

```
        return false
   }
   else
   {
        return true
   }
}

</SCRIPT>
<STYLE TYPE="TEXT/CSS">
   .menu  {display:none; margin-left:20px}
</STYLE>
</HEAD>
<BODY background="back1.gif">

<H3>
   <A HREF="completeinventory.htm"onClick="return toggleMenu('menu1')">Books
from the 1800s</A>
</H3>
<SPAN CLASS="menu" ID="menu1">
   1st Edition Charles Dicken's Tale of Two Cities<BR>
   1st Edition Edgar Allen Poe's The Tell Tale Heart<BR>
   Signed Copy of Edgar Allen Poe's The Cask of Amontillado<BR>

</SPAN>
<H3>
   <A HREF="completeinventory.htm" onClick="return toggleMenu('menu2')">Books
from the 1700's</A>
</H3>
<SPAN CLASS="menu" ID="menu2">
   Ben Franklin's Memoirs<BR>
   How to Start a Revolution<BR>
   Political Theory by Thomas Paine
</SPAN>
<H3>
   <A HREF="completeinventory.htm" onClick="return toggleMenu('menu3')">Books
for under $25</A>
</H3>
<SPAN CLASS="menu" ID="menu3">
   A Cowboy's Diary (circa 1850's)<BR>
   4th edition Charles Dicken's A Tale of Two Cities
   3rd Edition Jules Verne 20000 Leagues Under the Sea
</SPAN>

<H3>
```

```
   <A HREF="completeinventory.htm" onClick="return toggleMenu('menu4')">Our
finest items</A>
</H3>
<SPAN CLASS="menu" ID="menu4">
   Signed copy of Shakespeare's Hamlet<BR>
   Gutenberg's Diary<BR>
   Signed first Edition Longfellow
   1570 Hebrew Bible
</SPAN>

<CENTER><input type="submit" name="print" value="Print"onclick="window.print()">
</CENTER>
</BODY>
</HTML>
```

<completeinventory.htm>

Finishing the Pages

Now for the final two pages. The books.htm page gives us the abil ity to flip through the books that are cur rently being showcased. Com pute.htm gives us the abil ity to com pute costs.

```
<HTML>
<HEAD>
   <TITLE>Book Inventory</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT" >
   PicArray = new Array("book1.gif","book2.gif","book3.gif","book4.gif")
   CurrentPic = 0
   ImageCount = PicArray.length - 1

   function MovePrevious()
   {
        if (document.images && CurrentPic > 0)
        {
             CurrentPic–
             document.myPicture.src=PicArray[CurrentPic]
        }
   }

   function MoveNext()
   {
        if (document.images && CurrentPic < ImageCount )
        {
             CurrentPic++
             document.myPicture.src=PicArray[CurrentPic]
```

```
        }
   }

   </SCRIPT>
</HEAD>
<BODY background="back1.gif" >
<P>
<CENTER>
        <H2>This month's specials!</H2>
   <IMG SRC="book1.gif" HEIGHT=300 WIDTH =300 NAME="myPicture" ALT="Our Book
Inventory">
   <FORM>
          <INPUT TYPE="button" VALUE="<–" onClick="MovePrevious()">
          <INPUT TYPE ="button" VALUE ="–>" onClick="MoveNext()">
   </FORM>
</CENTER>
</BODY>
</HTML>
```

<books.htm>

```
<HTML>
<HEAD>
<TITLE>Complete Inventory</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT">

function toggleMenu(currMenu)
{
   if (document.getElementById)
   {
        thisMenu = document.getElementById(currMenu).style
        if (thisMenu.display == "block")
        {
             thisMenu.display = "none"
        }
        else
        {
             thisMenu.display = "block"
        }
        return false
   }
   else
   {
        return true
   }
}

</SCRIPT>
```

```
<STYLE TYPE="TEXT/CSS">
   .menu  {display:none; margin-left:20px}
</STYLE>
</HEAD>
<BODY background="back1.gif">

<H3>
   <A HREF="completeinventory.htm"onClick="return toggleMenu('menu1')">Books
from the 1800's</A>
</H3>
<SPAN CLASS="menu" ID="menu1">
   1st Edition Charles Dicken's Tale of Two Cities<BR>
   1st Edition Edgar Allen Poe's The Tell Tale Heart<BR>
   Signed Copy of Edgar Allen Poe's The Cask of Amontillado<BR>

</SPAN>
<H3>
   <A HREF="completeinventory.htm" onClick="return toggleMenu('menu2')">Books
from the 1700's</A>
</H3>
<SPAN CLASS="menu" ID="menu2">
   Ben Franklin's Memoirs<BR>
   How to Start a Revolution<BR>
   Political Theory by Thomas Paine
</SPAN>
<H3>
   <A HREF="completeinventory.htm" onClick="return toggleMenu('menu3')">Books
for under $25</A>
</H3>
<SPAN CLASS="menu" ID="menu3">
   A Cowboy's Diary (circa 1850's)<BR>
   4th edition Charles Dicken's A Tale of Two Cities
   3rd Edition Jules Verne 20000 Leagues Under the Sea
</SPAN>

<H3>
   <A HREF="completeinventory.htm" onClick="return toggleMenu('menu4')">Our
finest items</A>
</H3>
<SPAN CLASS="menu" ID="menu4">
   Signed copy of Shakespeare's Hamlet<BR>
   Gutenberg's Diary<BR>
   Signed first Edition Longfellow
   1570 Hebrew Bible
</SPAN>
```

```
<CENTER><input type="submit" name="print" value="Print"onclick="window.print()">
</CENTER>
</BODY>
</HTML>
```

<compute.htm>

Summary

The pur pose of this chap ter is bring together all of the work you have been doing. Here in one place is the com plete source code for the entire antique book store pro ject. All the required images are on the com pan ion CD. It is my sin cere wish that you have learned a lot while work ing your way through this book.

Appendix A

Online Resources

Below I have com piled a list of what I con sider to be the best JavaScript sites on the web. You can con sult these sites in order to get tuto ri als, source code, and much more. I strongly rec om mend that you famil iar ize your self with these web sites.

JavaScript Web Sites

JavaScript.com is one of the pre mier JavaScript sites. It has hun dreds of tuto ri-als and sam ples. You really want to use this one.
<http://www.javascript.com/>

Doc JavaScript is another excel lent site you would do well to ref er ence.
<http://www.webreference.com/js/>

A1 JavaScript is a site with some inter est ing source code you might wish to review.
<http://www.a1javascripts.com/>

JavaScript Gate is a pretty solid site with lots of source code and tuto ri als.
<http://javascriptgate.com/>

JavaScript World is, as the name implies, a com pi la tion of hun dreds of tuto ri als and source code.
<http://webdeveloper.com/>

JavaScript Games is a page with a num ber of games writ ten in JavaScript. It's a good page to look at to see what you can do with JavaScript.
<http://plaza.harmonix.ne.jp/~jimmeans/>

HTML Web Sites

HTML Ref er ence is a good site with a ref er ence to HTML 4.0 as well as JavaScript and other script ing lan guages.
http://www.geocities.com/pbb_webref/

The offi cial HTML 4.0 Spec i fi ca tion is an excel lent resource to find out exactly what is included in HTML 4.0.
<http://www.w3.org/TR/REC-html40/>

Organizational Web Sites

HTML Writers Guild is an asso ci a tion of HTML writ ers.
<http://www.hwg.org/>

Inter na tional WebMasters Asso ci a tion is an asso ci a tion of web mas ters and devel op ers. They have a vari ety of inter est ing mem ber ship ben e fits.
<http://www.irwa.org/>

Image Collections

When you are cre at ing websites, you need access to lots of images. The fol-low ing sites are my favor ites:

The Ani ma tion Fac tory at <http://www.animfactory.com> has a won der ful col-lec tion of ani mated gifs.

Gifs That Don't Suck at http://spit fire.cwv.net/~cdbailey/ is the place to find odd ani ma tions. You have to check this site out!

Certification Web Sites

If you are look ing to prove that you have learned JavaScript, BrainBench has a JavaScript cer tif i ca tion test you can take online:
www.brainbench.com <http://www.brainbench.com>

Employment Web Sites

If you are seek ing employ ment, there are sev eral websites you should check out:

Com puter Jobs.com www.computerjobs.com <http://www.computerjobs.com>

Jobs for Pro gram mers www.prgjobs.com <http://www.prgjobs.com>

Appendix B

HTML Reference

This is not meant to be a com pre hen sive HTML ref er ence. It is sim ply a ref er-ence for the essen tial HTML tags that you must know in order to con struct web pages.

Basic HTML Structure

<HTML> </HTML> Defines an HTML doc u ment

<HEAD> </HEAD> Defines the header sec tion

<TITLE>Page Title</TITLE> Defines the title that appears in the browser

<BODY> </BODY> Defines the body of the HTML doc u ment

<SCRIPT LANGUAGE = “JavaScript”> </SCRIPT> Defines a Java Script

Body Formatting Tags

<BODY BGCOLOR = blue> Sets the web page’s back ground color

<BODY BACKGROUND=”mypic.jpg”> Sets the back ground of the page to an image.

<BODY BGCOLOR = white TEXT= black LINK=blue VLINK=red ALINK=green> Sets the back ground color, text color, link color, vis ited link color, and active link color for the web page.

Images

To insert an image in your page you merely need to add the fol low ing:

<IMG SRC=”mypic.gif”>

Links

Text that is a web page link:

<A HREF = “http://www.wordware.com”> link to wordware pub lish ing</A>

An image that is a web page link:

<A HREF = “http://www.wordware.com”> <IMG SRC=”some pic-
ture.gif”</A>

Text that is an email link:

<A HREF=”mailto:chuckeasttom@yahoo.com”> email me!</A>

An image that is an email link:

<A HREF = “mailto:chuckeasttom@yahoo.com”> <IMG SRC =
“somepic.gif”> </A>

Lists

Unor dered list:

```
<UL>
   <LI> Item one
   <LI>Item two
</UL>
```

Ordered List:

```
<OL TYPE = I>
   <LI>
   <LI>
</OL>
```

Marquee

<MARQUEE BGCOLOR = yel low ALIGN Cen ter LOOP=infi nite>
This is a scroll ing Mar quee

</MARQUEE>

Tables

```
<TABLE BORDER =1>
   <TR>
      <TD> Row one cell 1
      <TD> Row one cell 2
   <TR>
      <TD> Row two cell 1
      <TD> Row two cell 2
</TABLE>
```

You can also set the back ground color of the table, a sin gle row, or even a sin-gle cell:

```
<TABLE BORDER =1 BGCOLOR = Blue>
<TR BGCOLOR = yel low>
 <TD BGCOLOR = green>
```

Text Formatting Tags

Italics <I> *this is ital ics* </I>

Bold <B> **this is bold** </B>

Under line <U> this is under lined</U>

Set ting font <FONT FACE = "Arial"> this is Arial font </FONT>

 this is a line break

<P> new para graph

Form Tags

<FORM> </FORM> defines an HTML form

<INPUT TYPE = text NAME = "txttest" VALUE = "Enter Text Here"> This defines a text field named 'txttest' with an ini tial value of "Enter Text Here."

<INPUT TYPE = but ton NAME = "Sub mit" VALUE = "OK" > This defines a but ton named Sub mit that has a cap tion of OK.

Appendix C

JavaScript Reference

abs Returns the abso lute (unsigned) value of its argu ment. Abs is a method of Math.

```
document.write(Math.abs(-5));
document.write(Math.abs(5))
```

These exam ples both return 5.

acos Returns the arc cosine (0 to pi radi ans) of its argu ment. The argu ment must be a num ber between –1 and 1. If the value is out side this range, a zero is returned. Method of Math. Also see asin, atan, cos, sin, tan.

alert Dis plays a JavaScript Alert dia log box with an OK but ton and a user-defined mes sage. See also METHODS con firm and prompt.

alinkColor The color of a link after the mouse but ton is depressed—but before it's released—and expressed as a hex a dec i mal RGB trip let or string lit eral.

appCodeName Returns a read-only string with the code name of the browser.

appName Returns a read-only string with the name of the browser. See also appCodeName, appVersion.

appVersion Returns a string with the ver sion infor ma tion of the browser.

```
document.write(navigator.appVersion)
```

returns

```
6.0 (Win95)
```

This spec i fies Nav i ga tor 6.0 run ning on Win dows 95 with an inter na tional release. See also appName, appCodeName.

asin Returns the arc sine (between –pi/2 and pi/2 radi ans) of a num ber between –1 and 1. If the num ber is out side this range, a zero is returned. Method of Math. See also acos, atan, cos, sin, tan.

atan Returns the arc tan gent (between –pi/2 and pi/2 radi ans) of a num ber between –1 and 1. If the num ber is out side this range, a zero is returned. Method of Math. See also acos, asin, cos, sin, tan.

back Recalls the pre vi ous URL from the his tory list. This method is the same as his tory.go(–1). Method of the His tory object. See also for ward and go.

bgColor The doc u ment back ground color expressed as a hex a dec i mal num ber or string lit eral. Prop erty of doc u ment. See also alinkColor, fgColor,linkColor, vlinkColor.

```
document.bgColor = "yellow"
```

blur Removes focus from the spec i fied form ele ment. For exam ple, the fol low-ing line removes focus from feed back:

```
sometext.blur()
```

assum ing that feed back is defined as:

```
<input type="text" name="sometext">
```

See also focus and select.

bold For mats a string object in bold text by encas ing it with HTML <B> tags. Method of string. See also ital ics, strike.

ceil Returns the small est inte ger greater than, or equal to, its argu ment. Method of Math. See also floor.

```
Math.ceil(3.05)
```

returns a 4.

charAt Returns the char ac ter from a string at the spec i fied index. The first char ac ter is at posi tion zero and the last at length –1. Method of string.

```
var userName = "Chuck Easttom"
document.write(userName.charAt(4)
```

returns a c.

See also indexOf and lastIndexOf.

checked A Boolean value (true or false) indi cat ing whether a check box or radio but ton is selected. The value is updated imme di ately when an item is checked. Prop erty of checkbox and radio.

clear Clears the con tents of a win dow, regard less of how the win dow was filled. Method of doc u ment. See also close, open, write, writeln.

close Closes the cur rent win dow. As with all win dow com mands, the win dow object is assumed.

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

both close the cur rent win dow.

See also clear, open, write, writeln.

confirm Dis plays a JavaScript con fir ma tion dia log box with a mes sage and buttons for OK and Can cel. Con firm returns a true if the user selects OK and false for Can cel.

See also alert and prompt.

cookie String value of a small piece of infor ma tion stored in a cli ent-side file. The value stored in the cookie is found using substring charAt, IndexOf, and lastIndexOf.

cos Returns the cosine of the argu ment. The angle must be in radi ans. Method of Math. See also acos, asin, atan, sin, tan.

fgColor The color of fore ground text rep re sented as a hex a dec i mal RGB trip let or a string lit eral. This value can not be changed after a doc u ment is loaded. Prop erty of doc u ment.

See also alinkColor, bgColor,linkColor, vlinkColor.

floor Returns the inte ger less than or equal to its argu ment. Method of Math.

```
Math.floor(5.78)
```

returns a 5.

See also ceil.

focus Nav i gates to a spe cific form ele ment and gives it focus.

fontcolor For mats the string object to a spe cific color expressed as a hex a dec i-mal RGB trip let or a string lit eral, sim i lar to using <font color=*color*>. Method of string.

fontsize For mats the string object to a spe cific font size: one of the seven defined sizes using an inte ger through the <fontsize=*size*> tag.

forward Loads the next doc u ment on the URL his tory list. This method is the same as his tory.go(1). Method of his tory. See also back and go.

getDate Returns the day of the month as an inte ger between 1 and 31. Method of Date. See also setDate.

getDay Returns the day of the week as an inte ger from zero (Sunday) to six (Sat ur day). Method of Date.

getHours Returns the hour of the day in 24-hour for mat, from zero (mid night) to 23 (11 PM). Method of Date. See also setHours.

getMinutes Returns the min utes with an inte ger from zero to 59. Method of Date. See also setMinutes.

getMonth Returns the month of the year as an inte ger between 0 (Jan u ary) and 11 (Decem ber). Method of Date. See also setMonth.

getSeconds Returns the sec onds in an inte ger from 0 to 59. Method of Date. See also setSeconds.

getYear Returns the year of the date object. Method of Date. See also setYear

go Loads a doc u ment spec i fied in the his tory list by its URL or rel a tive to the cur rent posi tion on the list. If the URL is incom plete, the clos est match is used. Method of his tory. See also back and for ward.

indexOf Returns the loca tion of a spe cific char ac ter or string, start ing the search from a spe cific loca tion. The first char ac ter of the string is spec i fied as zero and the last is the string's length–1.

Method of string. See also charAt and lastIndexof.

italics For mats a string object into ital ics by encas ing it an HTML <I> tag. Method of string. See also, bold, strike.

lastIndexOf Returns the index of a char ac ter or string in a string object by look ing back ward from the end of the string or a user-specified index. Method of string. See also charAt and indexOf.

linkColor The hyperlink color dis played in the doc u ment, expressed as a hex a-dec i mal RGB trip let or as a string lit eral. It works like the link attrib ute in the HTML <body> tag. Prop erty of doc u ment. See also alinkColor, bgColor, fgColor, vlinkColor.

open For a doc u ment, opens a stream to col lect the out put of write or writeln meth ods. If a doc u ment already exists in the tar get win dow, then the open method clears it.

pow Returns a base raised to an expo nent. Method of Math.

prompt Dis plays a prompt dia log box that accepts user input. If an ini tial value is not spec i fied for inputDefault, the dia log box dis plays the value <unde-fined>. Method of win dow. See also alert and con firm.

referrer Returns a read-only URL of the doc u ment that called the cur rent doc u-ment. It can be used to keep track of how users are linked to a page.

```
document.write("You came here from a page at " + document.referrer)
```

round Returns the value of a float ing-point argu ment rounded to the next high-est inte ger if the dec i mal por tion is greater than, or equal to, .5, or the next low est inte ger is less than .5. Method of Math.

setDate Sets the day of the month. Method of Date. See also getDate.

setHours Sets the hour for the cur rent time. Method of Date. See also getHours.

setMinutes Sets the min utes for the cur rent time. Method of Date. See also getMinutes.

setMonth Sets the month with an inte ger from 0 (Jan u ary) to 11 (Decem ber). Method of Date. See also getMonth.

setSeconds Sets the sec onds for the cur rent time. Method of Date. See also getSeconds.

setTime Sets the value of a date object. Method of Date. See also getTime.

setYear Sets the year in the cur rent date. See also getYear.

sin Returns the sine of an argu ment. The argu ment is the size of an angle expressed in radi ans, and the returned value is from –1 to 1. Method of Math. See also acos, asin, atan, cos, tan.

sqrt Returns the square root of a pos i tive numeric expres sion. If the argument's value is neg a tive, the returned value is zero.

tan Returns the tan gent of an argu ment. The argu ment is the size of an angle expressed in radi ans. Method of Math. See also acos, asin, atan, cos, sin.

toLowerCase Con verts all char ac ters in a string to low er case. Method of string. See also toUpperCase.

toUpperCase Con verts all char ac ters in a string to upper case. Method of string. See also toLowerCase.

vlinkColor Returns or sets the color of vis ited links.

write Writes one or more lines to a doc u ment win dow, and can include HTML tags and JavaScript expres sions, includ ing numeric, string, and log i cal val ues. The write method does not add a new line (
 or /n) char ac ter to the end of the out put. Method of doc u ment.

writeln Writes one or more lines to a doc u ment win dow fol lowed by a new line character.

Appendix D

Common Errors

Obvi ously there are a lot of pos si ble errors you could get in your code. My goal in this appen dix is to illus trate for you those errors that I find to be the most com mon. Hope fully if you will begin check ing for these errors, your debug ging pro cess will be much quicker.

Error	Example	Solution

Spelling	`Var myage`
	
	`Myag = 33`
		Always double-check your spelling

Missing Brackets	`<SCRIPT LANGUAGE = "JavaScript">`
	`Function squarenum()`
	`{`
	`  answer = num * num`
	
	`</SCRIPT>`
	
	Note: If you don't close the bracket, the browser will try to read the rest of the text as part of your function.
		Always make sure that any bracket you open, you also close.

Closing Tags	`<A HREF=http://wordware.com>`
	`WordWare publishing`
	
	Note: Since I omitted the </A>, the browser will attempt to execute the rest of the text as part of the hypertext reference.
		Many tags need to be closed. Make sure you close all tags that require closing.

Index