100%
ONE HUNDRED PERCENT
COMPREHENSIVE
AUTHORITATIVE
WHAT YOU NEED
ONE HUNDRED PERCENT

The perfect complement to your 4th or Gold Edition *JavaScript Bible*

Immerse yourself in a JavaScript master's code examples and analysis

Learn JavaScript and DOM behavior via interactive labs

Complete your *JavaScript Bible* library with this invaluable supplement

JavaScript™ Examples Bible

The Essential Companion to *JavaScript™ Bible*

Danny Goodman
Author of the bestselling *JavaScript Bible*

CD-ROM INSIDE!
Over 300 Ready-to-Run Example Scripts and More on CD-ROM!

Praise for Danny Goodman's *JavaScript Bible*

"*JavaScript Bible* is the definitive resource in JavaScript programming. I am never more than three feet from my copy."

—Steve Reich, CEO, PageCoders

"This book is a must-have for any Web developer or programmer."

—Thoma Lile, President, Kanis Technologies, Inc.

"Outstanding book. I would recommend this book to anyone interested in learning to develop advanced Web sites. Mr. Goodman did an excellent job of organizing this book and writing it so that even a beginning programmer can understand it."

—Jason Hensley, Director of Internet Services, NetVoice, Inc.

"Goodman is always great at delivering clear and concise technical books!"

—Dwayne King, Chief Technology Officer, White Horse

"*JavaScript Bible* is well worth the money spent!"

— Yen C.Y. Leong, IT Director, Moo Mooltimedia, a member of SmartTransact Group

"A must-have book for any Internet developer."

— Uri Fremder, Senior Consultant, TopTier Software

"I love this book! I use it all the time, and it always delivers. It's the only JavaScript book I use!"

—Jason Badger, Web Developer

"Whether you are a professional or a beginner, this is a great book to get."

—Brant Mutch, Web Application Developer, Wells Fargo Card Services, Inc.

"I never thought I'd ever teach programming before reading your book [*JavaScript Bible*]. It's so simple to use—the Programming Fundamentals section brought it all back! Thank you for such a wonderful book, and for breaking through my programming block!"

—Susan Sann Mahon, Certified Lotus Instructor, TechNet Training

"I continue to get so much benefit from *JavaScript Bible*. What an amazing book! Danny Goodman is the greatest!"

—Patrick Moss

"Danny Goodman is very good at leading the reader into the subject. *JavaScript Bible* has everything we could possibly need."

—Philip Gurdon

"An excellent book that builds solidly from whatever level the reader is at. A book that is both witty and educational."

—*Dave Vane*

"I continue to use the book on a daily basis and would be lost without it."

—*Mike Warner, Founder, Oak Place Productions*

"*JavaScript Bible* is by *far* the best JavaScript resource I've ever seen (and I've seen quite a few)."

—*Robert J. Mirro, Independent Consultant, RJM Consulting*

JavaScript™ Examples Bible: The Essential Companion to *JavaScript™ Bible*

JavaScript™ Examples Bible: The Essential Companion to *JavaScript™ Bible*

Danny Goodman

Hungry Minds™

Best-Selling Books • Digital Downloads • e-Books • Answer Networks • e-Newsletters • Branded Web Sites • e-Learning

Indianapolis, IN ✦ Cleveland, OH ✦ New York, NY

JavaScript™ Examples Bible: The Essential Companion to *JavaScript™ Bible*

Published by:
Hungry Minds, Inc.
909 Third Avenue
New York, NY 10022
www.hungryminds.com

Library of Congress Control No.: 2001091964

ISBN: 0-7645-4855-7

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/RY/QX/QR/IN

Distributed in the United States by Hungry Minds, Inc.

Distributed by CDG Books Canada Inc. for Canada; by Transworld Publishers Limited in the United Kingdom; by IDG Norge Books for Norway; by IDG Sweden Books for Sweden; by IDG Books Australia Publishing Corporation Pty. Ltd. for Australia and New Zealand; by TransQuest Publishers Pte Ltd. for Singapore, Malaysia, Thailand, Indonesia, and Hong Kong; by Gotop Information Inc. for Taiwan; by ICG Muse, Inc. for Japan; by Intersoft for South Africa; by Eyrolles for France; by International Thomson Publishing for Germany, Austria, and Switzerland; by Distribuidora Cuspide for Argentina; by LR International for Brazil; by Galileo Libros for Chile; by Ediciones ZETA S.C.R. Ltda. for Peru; by WS Computer Publishing Corporation, Inc., for the Philippines; by Contemporanea de Ediciones for Venezuela; by Express Computer Distributors for the Caribbean and West Indies; by Micronesia Media Distributor, Inc. for Micronesia; by Chips Computadoras S.A. de C.V. for Mexico; by Editorial Norma de Panama S.A. for Panama; by American Bookshops for Finland.

For general information on Hungry Minds' products and services please contact our Customer Care department; within the U.S. at 800-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

For sales inquiries and resellers information, including discounts, premium and bulk quantity sales and foreign language translations please contact our Customer Care department at 800-434-3422, fax 317-572-4002 or write to Hungry Minds, Inc., Attn: Customer Care department, 10475 Crosspoint Boulevard, Indianapolis, IN 46256.

For information on licensing foreign or domestic rights, please contact our Sub-Rights Customer Care department at 212-884-5000.

For information on using Hungry Minds' products and services in the classroom or for ordering examination copies, please contact our Educational Sales department at 800-434-2086 or fax 317-572-4005.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 317-572-3168 or fax 317-572-4168.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 978-750-4470.

About the Author

Danny Goodman is the author of numerous critically acclaimed and bestselling books, including *The Complete HyperCard Handbook*, *Danny Goodman's AppleScript Handbook*, and *Dynamic HTML: The Definitive Reference*. He is a renowned authority and expert teacher of computer scripting languages and is widely known for his "JavaScript Apostle" articles in Netscape's *ViewSource* online developer newsletter. His writing style and pedagogy continue to earn praise from readers and teachers around the world. To help keep his finger on the pulse of real-world programming challenges, Goodman frequently lends his touch as consulting programmer and designer to leading-edge World Wide Web and intranet sites from his home base in the San Francisco area.

Credits

Acquisitions Editor
Sharon Cox

Project Editor
Neil Romanosky

Technical Editor
David Wall

Copy Editors
Jerelind Charles
Victoria Lee O'Malley

Editorial Manager
Colleen Totz

Project Coordinator
Regina Snyder

Graphics and Production Specialists
Gabriele McCann
Betty Schulte
Jeremey Unger
Erin Zeltner

Quality Control Technicians
Laura Albert
David Faust
Andy Hollandbeck

Permissions Editor
Laura Moss

Media Development Specialist
Greg Stephens

Media Development Coordinator
Marisa Pearman

Book Designer
Kurt Krames

Proofreading and Indexing
TECHBOOKS Production Services

Cover Illustrator
Kate Shaw

Preface

A common thread running throughout most of my computer-book–writing career is that I tend to write a book I wish I had had in order to learn a new technology in the first place. Because I must write that book without the benefit of existing models, I begin by doing my best to master the technology, and then I write the book to help other newcomers learn as much as I did, but more quickly and with less pain, anguish, and confusion. To accomplish that goal, I write as much content as I feel is necessary to cover the topic in the depth that my readers require.

When I started on what became the 4th and Gold editions of the *JavaScript Bible*, there were models to follow (my previous three editions) plus a substantial amount of brand new material, much of which had not yet been documented anywhere. I also assumed the responsibility of integrating the frequently conflicting and competing philosophies of the ways the JavaScript language is applied to a variety of browser brands and versions. Resolving these conflicts is a challenge that I face in my own programming work with clients, and I take great pleasure in sharing my solutions and approaches with other programmers floating in the same boat.

As my editor and I began counting the pages I had assembled for these new editions, we discovered that the number of pages far outstripped the printer's binding capabilities, even in a thicker volume made possible by using a hard cover (the Gold edition). Certainly not all of the words that I had written were so precious that some of them couldn't be cut. But we were hundreds of pages beyond capacity. To cut that much content would have forced exclusion of coverage of language or document object model vocabulary.

Fortunately, as had been done in previous editions, the plan for the new editions included Adobe Acrobat versions of the books on the accompanying CD-ROM. Although a significant compromise to ease of reading, it was possible to move some of the book's content to the CD-ROM and leave the most important parts on the printed page. For the softcover 4th edition, reference chapters covering less-used or advanced subjects were pulled from print; for the hardcover Gold edition, which was longer and targeted more for professional scripters, the advanced chapters were put back into the book (along with 15 additional chapters for that edition), and the JavaScript tutorial was exiled to the CD-ROM.

But even after making the difficult decisions about which chapters could go to the CD-ROMs, the page counts for both volumes were still excessive. Something else — something big — had to go. The remaining bundle that could free us from the page

count devil was all of the Example sections from the reference vocabulary. By being nondiscriminatory about these extractions — that is, extracting all of them instead of only selected sections — we could convey to readers a consistent organizational model.

In the end, the extracted Example sections from Parts III and IV found their way into Appendix F on the CD-ROMs of both editions of the larger tome. I knew that as a reader of my own books (and one of a certain age at that) I would not enjoy having to flip back and forth between book and screen to refresh my memory about a term and see it in action. A more pleasing solution for many *JavaScript Bible* readers would be a separate volume containing a printed version of the Examples sections. The new volume would act as a companion to both the 4th and Gold editions of the *JavaScript Bible*.

Using Appendix F as a starting point, I divided the content into chapters along the same lines as the *JavaScript Bible* reference sections. This also gave me a chance to study the examples for each chapter with fresh eyes. The examples haven't changed, but I had the opportunity to direct the reader's attention to examples that I thought were particularly helpful in mastering a document-level or core language object. Thus, each chapter of this book begins with a scene-setting introduction and a list of highlights to which you should pay special attention. Also, since you will likely be scanning through the book from time to time, I added many illustrations of the pages produced from the code listings. These figures will help you visualize what important listing code does when the page is loaded into a browser.

Now you know the story behind the *JavaScript Examples Bible*. Some budget-conscious readers may not be thrilled to pay more for what appears to be a printout of content they already own in electronic format. If so, then please continue using the Acrobat version. But if, like me, you enjoy the portability and visual scanability of a printed work, then keeping this book near your *JavaScript Bible* volume will enhance your learning and research activities.

Organization and Features of This Book

Almost all chapters in this book correspond to similarly named chapters in Parts III and IV from the *JavaScript Bible* 4th and Gold editions. Although chapters in this book are consecutively numbered starting with Chapter 1, each chapter title includes a reference to the corresponding chapter number from the big books. For example, Chapter 1 of this book provides the Examples sections for terms related to generic HTML elements. That subject is covered in Chapter 15 of the big books. There is not always a one-to-one relationship between chapters. Several chapters of the big books have no Examples sections in them because sample code is embedded as part of the big book text. Therefore, don't be surprised to see gaps in pointers to *JavaScript Bible* reference chapters.

Listing numbers are derived from their original order in what had been planned as a contiguous volume. Such listing numbers are the ones referred to in the "On the CD-ROM" pointers throughout Parts III and IV of the big books. This should help you locate an example's listing when you reach one of those pointers in the *JavaScript Bible*. Notice, too, that the big books' running footers with property, method, and event handler names appear in this book, too. Therefore, if you should be looking at an example listing of this book and wish to consult the more detailed discussion of the subject in the large book, turn to the corresponding big book chapter and locate the corresponding terminology within the object's chapter.

Many examples throughout this book refer to The Evaluator. This Web page application is described at length in Chapter 13 of the big books. You can find the file for The Evaluator within the Listings\Chap13 folder on the CD-ROM for either the big book or this book.

CD-ROM

The accompanying CD-ROM contains the complete set of over 300 ready-to-run HTML documents from the *JavaScript Bible, Gold Edition*. These include listings for both the Examples sections in this book and all other listings from the Gold edition. You can run these examples with your JavaScript-enabled browser, but be sure to use the `index.html` page in the Listings folder as a gateway to running the listings. This page shows you the browsers that are compatible with each example listing.

The Quick Reference from Appendix A of the big books is in `.pdf` format on the CD-ROM for you to print out and assemble as a handy reference, if desired. Adobe Acrobat Reader is included on the CD-ROM so that you can read this `.pdf` file. Finally, the text of the book is in a `.pdf` file format on the CD-ROM for easy searching.

Formatting and Naming Conventions

The script listings and words in this book are presented in a `monospace font` to set them apart from the rest of the text. Because of restrictions in page width, lines of script listings may, from time to time, break unnaturally. In such cases, the remainder of the script appears in the following line, flush with the left margin of the listing, just as they would appear in a text editor with word wrapping turned on. If these line breaks cause you problems when you type a script listing into a document yourself, I encourage you to access the corresponding listing on the CD-ROM to see how it should look when you type it.

To make it easier to spot in the text when a particular browser and browser version is required, most browser references consist of a two-letter abbreviation and a version number. For example, IE5 means Internet Explorer 5 for any operating system;

NN6 means Netscape Navigator 6 for any operating system. If a feature is introduced with a particular version of browser and is supported in subsequent versions, a plus symbol (+) follows the number. For example, a feature marked IE4+ indicates that Internet Explorer 4 is required at a minimum, but the feature is also available in IE5, IE5.5, and so on. Occasionally, a feature or some highlighted behavior applies to only one operating system. For example, a feature marked IE4+/Windows means that it works only on Windows versions of Internet Explorer 4 or later. As points of reference, the first scriptable browsers were NN2, IE3/Windows, and IE3.01/Macintosh. Moreover, IE3 for Windows can be equipped with one of two versions of the `JScript.dll` file. A reference to the earlier version is cited as IE3/J1, while the later version is cited as IE3/J2. You will see this notation primarily in the compatibility charts throughout the reference chapters.

Acknowledgments

Because most of the content of this volume was created as part of the *JavaScript Bible*, the acknowledgments that you see in your copy of the 4th or Gold editions apply equally to this volume. But this *JavaScript Examples Bible* did not come into being without additional effort on the part of dedicated Hungry Minds, Inc., staff. In particular, I want to thank Sharon Cox for turning my idea into a title, and editor Neil Romanosky, who, even after marshaling over 4,000 pages of content for the 4th and Gold editions, took charge of this volume to maintain continuity across the entire series. Thanks, too, to my friends and family, who certainly must have grown weary of my tales of reaching schedule milestones on this project not once, not twice, but three times over many, many months.

Contents at a Glance

Contents

CHAPTER 1

Generic HTML Element Objects (Chapter 15)

In This Chapter

Understanding element containment relationships

Common properties and methods of all HTML element objects

Event handlers of all element objects

Document object models for both IE4+ and NN6 expose all HTML elements as scriptable objects. A beneficial byproduct of this concept is that object model designers find it easier to implement their models according to genuinely object-oriented principles. (In truth, modern HTML and DOM industry standards encourage browser makers to think in object-oriented terms anyway.) The object-oriented principle most applicable to the way we work with objects is that all HTML elements inherit properties, methods, and event handlers from a generic (and unseen) HTML element object. Thus, specifications for any HTML element object start with those of the generic object, and then pile on element-specific features, such as the `src` property of an IMG element. This chapter deals almost exclusively with the properties, methods, and event handlers that all HTML elements have in common.

Examples Highlights

- Modern object models and the scripting world now pay much attention to the containment hierarchy of elements and text nodes in a document. The function shown in Listing 15-3 demonstrates how vital the `childNodes` property is to scripts that need to inspect (and then perhaps modify) page content.
- Element containment is also at the forefront in Listing 15-10, where W3C DOM syntax demonstrates how to use the `firstChild` and `lastChild` properties, plus the `insertBefore()`, `appendChild()`, and `replaceChild()` methods, to change portions of page content on the fly.
- In the IE/Windows world, data binding can be a powerful tool that requires only tiny amounts of your code in a page. You can get a good sense of the possibilities in the extended examples for the `dataFld` and related properties.

- Follow the steps for the `disabled` property to see how form controls can be disabled in IE4+ and NN6. IE5.5 lets you disable any element on the page, as you can witness in real time when you follow the example steps.
- Long-time IE scripters know the powers of the `innerHTML` and `innerText` properties. Listing 15-11 solidifies by example the precise differences between the two related properties. Only one of these properties, `innerHTML`, is implemented in NN6.
- Grasping the details of properties that govern element positions and dimensions is not easy, as noted in the *JavaScript Bible* text. But you can work through the examples of the client-, offset-, and scroll-related properties for IE4+ and the offset-related properties in NN6 to help you visualize what these properties control. If you are scripting cross-browser applications, be sure to work through the offset-related properties in both browsers to compare the results.
- Compare the IE5+ `attachEvent()` method and NN6 `addEventListener()` method for modern ways to assign event handlers to element objects. Although the method names are different, the two work identically.
- Observe how the `getAttribute()` method returns an object's property value when the property name is a string and the name is the same as an assigned element attribute name. The `getAttribute()` method is the prescribed way to retrieve property values according to the W3C DOM.
- You can see how the `getElementsByTagName()` method returns an array of nested elements with a particular tag. This is a great way, for example, to get a one-dimensional array of all cells within a table.
- Spend time comparing how the various insert- and replace-related methods operate from different points of view. In the IE world, most operate on the current element; in the W3C DOM world, the methods operate on child nodes of the current element.
- For IE5+/Windows, check out the way dynamic properties are managed through the `getExpression()`, `setExpression()`, and `recalc()` methods. Listing 15-32 demonstrates a neat graphical clock that employs these methods.
- IE5+/Windows provides a number of event handlers, such as `onBeforeCopy`, `onBeforePaste`, `onCopy`, `onCut`, and `onPaste` that let scripts manage the specific information preserved in the clipboard. These event handlers can also be used with the `onContextMenu` event handler to facilitate custom context menus.
- Another set of IE5+/Windows event handlers provides excellent control over user dragging and dropping of elements on a page. Listing 15-37 is particularly interesting in this regard.
- Listing 15-41 shows a cross-browser laboratory for understanding the three keyboard events and how to get key and character information from the event. You see event-handling that works with IE4+, NN4, and NN6 event models.
- Numerous mouse-related events belong to all HTML elements. Listings 15-42 and 15-43 demonstrate simplified image swapping and element dragging.

Generic Objects

Properties

accessKey

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

When you load the script in Listing 15-1, adjust the height of the browser window so that you can see nothing below the second dividing rule. Enter any character into the Settings portion of the page and press Enter. (The Enter key may cause your computer to beep.) Then hold down the Alt (Windows) or Ctrl (Mac) key while pressing the same keyboard key. The element from below the second divider should come into view.

Listing 15-1: Controlling the accessKey Property

```
<HTML>
<HEAD>
<TITLE>accessKey Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function assignKey(type, elem) {
    if (window.event.keyCode == 13) {
        switch (type) {
            case "button":
                document.forms["output"].access1.accessKey = elem.value
                break
            case "text":
                document.forms["output"].access2.accessKey = elem.value
                break
            case "table":
                document.all.myTable.accessKey = elem.value
        }
        return false
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>accessKey Property Lab</H1>
<HR>
Settings:<BR>
```

Continued

Listing 15-1 *(continued)*

```
<FORM NAME="input">
Assign an accessKey value to the Button below and press Return:
<INPUT TYPE="text" SIZE=2 MAXLENGTH=1
onKeyPress="return assignKey('button', this)">
<BR>
Assign an accessKey value to the Text Box below and press Return:
<INPUT TYPE="text" SIZE=2 MAXLENGTH=1
onKeyPress="return assignKey('text', this)">
<BR>
Assign an accessKey value to the Table below (IE5.5 only) and press Return:
<INPUT TYPE="text" SIZE=2 MAXLENGTH=1
onKeyPress="return assignKey('table', this)">
</FORM>
<BR>
Then press Alt (Windows) or Control (Mac) + the key.
<BR>
<I>Size the browser window to view nothing lower than this line.</I>
<HR>

<FORM NAME="output" onSubmit="return false">
<INPUT TYPE="button" NAME="access1" VALUE="Standard Button">
<P></P>
<INPUT TYPE="text" NAME="access2">
<P></P>
</FORM>
<TABLE ID="myTable" CELLPADDING="10" BORDER=2>
<TR>
<TH>Quantity<TH>Description<TH>Price
</TR>
<TBODY BGCOLOR="red">
<TR>
    <TD WIDTH=100>4<TD>Primary Widget<TD>$14.96
</TR>
<TR>
    <TD>10<TD>Secondary Widget<TD>$114.96
</TR>
</TBODY>
</TABLE>

</BODY>
</HTML>
```

In IE5, the keyboard combination may bring focus to the input field. This anomalous behavior does not affect the normal script setting of the `accessKey` property.

all

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `all` collection. Enter the following statements one at a time into the lower text box, and review the results in the textarea for each.

```
document.all
myTable.all
myP.all
```

If you encounter a numbered element within a collection, you can explore that element to see which tag is associated with it. For example, if one of the results for the `document.all` collection says `document.all.8=[object]`, enter the following statement into the topmost text box:

```
document.all[8].tagName
```

attributes

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to examine the values of the `attributes` array for some of the elements in that document. Enter each of the following expressions into the lower text field, and see the array contents in the Results textarea for each:

```
document.body.attributes
document.getElementById("myP").attributes
document.getElementById("myTable").attributes
```

If you have both NN6 and IE5, compare the results you get for each of these expressions. To view the properties of a single attribute in IE5/Windows, enter the following statement into the bottom text field:

```
document.getElementById("myP").attributes["class"]
```

For NN6 and IE5/Mac, use the W3C DOM syntax:

```
document.getElementById("myP").attributes.getNamedItem("class")
```

behaviorUrns

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

The following function is embedded within a more complete example of IE/Windows HTML behaviors (Listing 15-19 in this chapter). It reports the length of the `behaviorUrns` array and shows — if the values are returned — the URL of the attached behavior.

```
function showBehaviors() {
    var num = document.all.myP.behaviorUrns.length
    var msg = "The myP element has " + num + " behavior(s). "
    if (num > 0) {
        msg += "Name(s): \r\n"
        for (var i = 0; i < num; i++) {
            msg += document.all.myP.behaviorUrns[i] + "\r\n"
        }
    }
    alert(msg)
}
```

canHaveChildren

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-2 uses color to demonstrate the difference between an element that can have children and one that cannot. The first button sets the `color` style property of every visible element on the page to red. Thus, elements (including the normally non-childbearing ones such as HR and INPUT) are affected by the color change. But if you reset the page and click the largest button, only those elements that can contain nested elements receive the color change.

Listing 15-2: **Reading the canHaveChildren Property**

```
<HTML>
<HEAD>
<TITLE>canHaveChildren Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
```

```
function colorAll() {
    for (var i = 0; i < document.all.length; i++) {
        document.all[i].style.color = "red"
    }
}

function colorChildBearing() {
    for (var i = 0; i < document.all.length; i++) {
        if (document.all[i].canHaveChildren) {
            document.all[i].style.color = "red"
        }
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>canHaveChildren Property Lab</H1>
<HR>
<FORM NAME="input">
<INPUT TYPE="button" VALUE="Color All Elements" onClick="colorAll()">
<BR>
<INPUT TYPE="button" VALUE="Reset" onClick="history.go(0)">
<BR>
<INPUT TYPE="button" VALUE="Color Only Elements That Can Have Children"
onClick="colorChildBearing()">
</FORM>
<BR>
<HR>

<FORM NAME="output">
<INPUT TYPE="checkbox" CHECKED>Your basic checkbox
<P></P>
<INPUT TYPE="text" NAME="access2" VALUE="Some textbox text.">
<P></P>
</FORM>
<TABLE ID="myTable" CELLPADDING="10" BORDER=2>
<TR>
<TH>Quantity<TH>Description<TH>Price
</TR>
<TBODY>
<TR>
    <TD WIDTH=100>4<TD>Primary Widget<TD>$14.96
</TR>
<TR>
    <TD>10<TD>Secondary Widget<TD>$114.96
</TR>
</TBODY>
</TABLE>

</BODY>
</HTML>
```

canHaveHTML

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `canHaveHTML` property. Enter the following statements into the top text field and observe the results:

```
document.all.input.canHaveHTML
document.all.myP.canHaveHTML
```

The first statement returns `false` because an INPUT element (the top text field in this case) cannot have nested HTML. But the `myP` element is a P element that gladly accepts HTML content.

childNodes

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	

Example

The `walkChildNodes()` function shown in Listing 15-3 accumulates and returns a hierarchical list of child nodes from the point of view of the document's HTML element (the default) or any element whose ID you pass as a string parameter. This function is embedded in The Evaluator so that you can inspect the child node hierarchy of that page or (when using `evaluator.js` for debugging as described in Chapter 45 of the *JavaScript Bible*) the node hierarchy within any page you have under construction. Try it out in The Evaluator by entering the following statements into the top text field:

```
walkChildNodes()
walkChildNodes(getElementById("myP"))
```

The results of this function show the nesting relationships among all child nodes within the scope of the initial object. It also shows the act of drilling down to further `childNodes` collections until all child nodes are exposed and catalogued. Text nodes are labeled accordingly. The first 15 characters of the actual text are placed in the results to help you identify the nodes when you compare the results against your HTML source code. The early NN6 phantom text nodes that contain carriage returns display `<cr>` in the results for each return character.

Listing 15-3: **Collecting Child Nodes**

```
function walkChildNodes(objRef, n) {
    var obj
    if (objRef) {
        if (typeof objRef == "string") {
            obj = document.getElementById(objRef)
        } else {
            obj = objRef
        }
    } else {
        obj = (document.body.parentElement) ?
            document.body.parentElement : document.body.parentNode
    }
    var output = ""
    var indent = ""
    var i, group, txt
    if (n) {
        for (i = 0; i < n; i++) {
            indent += "+---"
        }
    } else {
        n = 0
        output += "Child Nodes of <" + obj.tagName
        output += ">\n====================\n"
    }
    group = obj.childNodes
    for (i = 0; i < group.length; i++) {
        output += indent
        switch (group[i].nodeType) {
            case 1:
                output += "<" + group[i].tagName
                output += (group[i].id) ? " ID=" + group[i].id : ""
                output += (group[i].name) ? " NAME=" + group[i].name : ""
                output += ">\n"
                break
            case 3:
                txt = group[i].nodeValue.substr(0,15)
                output += "[Text:\"" + txt.replace(/[\r\n]/g,"<cr>")
                if (group[i].nodeValue.length > 15) {
                    output += "..."
                }
                output += "\"]\n"
                break
            case 8:
                output += "[!COMMENT!]\n"
                break
            default:
                output += "[Node Type = " + group[i].nodeType + "]\n"
        }
```

Continued

Listing 15-3 *(continued)*

```
        if (group[i].childNodes.length > 0) {
            output += walkChildNodes(group[i], n+1)
        }
    }
    return output
}
```

children

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The `walkChildren()` function in Listing 15-4 accumulates and returns a hierarchical list of child elements from the point of view of the document's HTML element (the default) or any element whose ID you pass as a string parameter. This function is embedded in The Evaluator so that you can inspect the parent–child hierarchy of that page or (when using `evaluator.js` for debugging, as described in Chapter 45 of the *JavaScript Bible*) the element hierarchy within any page you have under construction. Try it out in The Evaluator in IE5+ by entering the following statements into the top text field:

```
walkChildren()
walkChildren("myP")
```

The results of this function show the nesting relationships among all parent and child elements within the scope of the initial object. It also shows the act of drilling down to further `children` collections until all child elements are exposed and catalogued. The element tags also display their `ID` and/or `NAME` attribute values if any are assigned to the elements in the HTML source code.

Listing 15-4: **Collecting Child Elements**

```
function walkChildren(objRef, n) {
    var obj
    if (objRef) {
        if (typeof objRef == "string") {
            obj = document.getElementById(objRef)
        } else {
            obj = objRef
```

```
        }
    } else {
        obj = document.body.parentElement
    }
    var output = ""
    var indent = ""
    var i, group
    if (n) {
        for (i = 0; i < n; i++) {
            indent += "+---"
        }
    } else {
        n = 0
        output += "Children of <" + obj.tagName
        output += ">\n====================\n"
    }
    group = obj.children
    for (i = 0; i < group.length; i++) {
        output += indent + "<" + group[i].tagName
        output += (group[i].id) ? " ID=" + group[i].id : ""
        output += (group[i].name) ? " NAME=" + group[i].name : ""
        output += ">\n"
        if (group[i].children.length > 0) {
            output += walkChildren(group[i], n+1)
        }
    }
    return output
}
```

`className`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The style of an element toggles between "on" and "off" in Listing 15-5 by virtue of setting the element's `className` property alternatively to an existing style sheet class selector name and an empty string. When you set the `className` to an empty string, the default behavior of the H1 element governs the display of the first header. A click of the button forces the style sheet rule to override the default behavior in the first H1 element.

Listing 15-5: **Working with the className Property**

```
<HTML>
<HEAD>
<TITLE>className Property</TITLE>
<STYLE TYPE="text/css">
.special {font-size:16pt; color:red}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function toggleSpecialStyle(elemID) {
    var elem = (document.all) ? document.all(elemID) :
document.getElementById(elemID)
    if (elem.className == "") {
        elem.className = "special"
    } else {
        elem.className = ""
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>className Property Lab</H1>
<HR>
<FORM NAME="input">
<INPUT TYPE="button" VALUE="Toggle Class Name"
onClick="toggleSpecialStyle('head1')">
</FORM>
<BR>
<H1 ID="head1">ARTICLE I</H1>
<P>Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of speech, or of
the press; or the right of the people peaceably to assemble, and to petition the
government for a redress of grievances.</P>

<H1>ARTICLE II</H1>
<P>A well regulated militia, being necessary to the security of a free state,
the right of the people to keep and bear arms, shall not be infringed.</P>
</BODY>
</HTML>
```

You can also create multiple versions of a style rule with different class selector identifiers and apply them at will to a given element.

clientHeight
clientWidth

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 15-6 calls upon the `clientHeight` and `clientWidth` properties of a DIV element that contains a paragraph element. Only the width of the DIV element is specified in its style sheet rule, which means that the paragraph's text wraps inside that width and extends as deeply as necessary to show the entire paragraph. The `clientHeight` property describes that depth. The `clientHeight` property then calculates where a logo image should be positioned immediately after DIV, regardless of the length of the text. As a bonus, the `clientWidth` property helps the script center the image horizontally with respect to the paragraph's text.

Listing 15-6: **Using clientHeight and clientWidth Properties**

```
<HTML>
<HEAD>
<TITLE>clientHeight and clientWidth Properties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function showLogo() {
    var paragraphW = document.all.myDIV.clientWidth
    var paragraphH = document.all.myDIV.clientHeight
    // correct for Windows/Mac discrepancies
    var paragraphTop = (document.all.myDIV.clientTop) ?
        document.all.myDIV.clientTop : document.all.myDIV.offsetTop
    var logoW = document.all.logo.style.pixelWidth
    // center logo horizontally against paragraph
    document.all.logo.style.pixelLeft = (paragraphW-logoW)/2
    // position image immediately below end of paragraph
    document.all.logo.style.pixelTop = paragraphTop + paragraphH
    document.all.logo.style.visibility = "visible"
}
</SCRIPT>
</HEAD>
<BODY>
<BUTTON onClick="showLogo()">Position and Show Logo Art</BUTTON>
<DIV ID="logo" STYLE="position:absolute; width:120px; visibility:hidden"><IMG
SRC="logo.gif"></DIV>
<DIV ID="myDIV" STYLE="width:200px">
```

Continued

Listing 15-6 *(continued)*

```
<P>Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim adminim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit involuptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.</P>
</DIV>
</BODY>
</HTML>
```

To assist in the vertical positioning of the logo, the `offsetTop` property of the DIV object provides the position of the start of the DIV with respect to its outer container (the BODY). Unfortunately, IE/Mac uses the `clientTop` property to obtain the desired dimension. That measure (assigned to the `paragraphTop` variable), plus the `clientHeight` of the DIV, provides the top coordinate of the image.

If you use only IE5, you can eliminate the DIV wrapper around the P element and assign the `STYLE` attribute directly to the P element. The script can then read the `clientHeight` and `clientWidth` of the P object.

contentEditable

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Listing 15-7 is a simplified demonstration of how to turn some text inside a document into an editable element. When you click the button of a freshly loaded page, the `toggleEdit()` function captures the opposite of the current editable state via the `isContentEditable` property of the DIV that is subject to edit. You switch on editing for that element in the next statement by assigning the new value to the `contentEditable` property of the DIV. For added impact, turn the text of the DIV to red to provide additional user feedback about what is editable on the page. You can also switch the button label to one that indicates the action invoked by the next click on it.

Listing 15-7: **Using the contentEditable Property**

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
.normal {color: black}
.editing {color: red}
```

```
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function toggleEdit() {
    var newState = !editableText.isContentEditable
    editableText.contentEditable = newState
    editableText.className = (newState) ? "editing" : "normal"
    editBtn.innerText = (newState) ? "Disable Editing" : "Enable Editing"
}
</SCRIPT>
<BODY>
<H1>Editing Contents</H1>
<HR>
<P>Turn on editing to modify the following text:</P>
<DIV ID="editableText">Edit this text on the fly....</DIV>
<P><BUTTON ID="editBtn" onClick="toggleEdit()" onFocus="this.blur()">
Enable Editing
</BUTTON></P>
</BODY>
</HTML>
```

The BUTTON element has an `onFocus` event handler that immediately invokes the `blur()` method on the button. This prevents a press of the spacebar (during editing) from accidentally triggering the button.

currentStyle

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to compare the properties of the `currentStyle` and `style` objects of an element. For example, an unmodified copy of The Evaluator contains an EM element whose ID is `"myEM"`. Enter `document.all.myEM.style` into the bottom property listing text box and press Enter. Notice how most of the property values are empty. Now enter `document.all.myEM.currentStyle` into the property listing text box and press Enter. Every property has a value associated with it.

dataFld
dataFormatAs
dataSrc

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 15-8 is a simple document that has two TDC objects associated with it. The external files are different formats of the U.S. Bill of Rights document. One file is a traditional, tab-delimited data file consisting of only two records. The first record is a tab-delimited sequence of field names (named `"Article1"`, `"Article2"`, and so on); the second record is a tab-delimited sequence of article content defined in HTML:

```
<H1>ARTICLE I</H1><P>Congress shall make...</P>
```

The second file is a raw text file consisting of the full Bill of Rights with no HTML formatting attached.

When you load Listing 15-8, only the first article of the Bill of Rights appears in a blue-bordered box. Buttons enable you to navigate to the previous and next articles in the series. Because the data source is a traditional, tab-delimited file, the `nextField()` and `prevField()` functions calculate the name of the next source field and assign the new value to the `dataFld` property. All of the data is already in the browser after the page loads, so cycling through the records is as fast as the browser can reflow the page to accommodate the new content.

Listing 15-8: **Changing dataFld and dataSrc Properties**

```
<HTML>
<HEAD>
<TITLE>Data Binding</TITLE>
<STYLE TYPE="text/css">
#display {width:500px; border:10px ridge blue; padding:20px}
.hiddenControl {display:none}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function nextField() {
    var elem = document.all.display
    var fieldName = elem.dataFld
    var currFieldNum = parseInt(fieldName.substring(7, fieldName.length),10)
    currFieldNum = (currFieldNum == 10) ? 1 : ++currFieldNum
    elem.dataFld = "Article" + currFieldNum
}
```

```
function prevField() {
    var elem = document.all.display
    var fieldName = elem.dataFld
    var currFieldNum = parseInt(fieldName.substring(7, fieldName.length),10)
    currFieldNum = (currFieldNum == 1) ? 10 : --currFieldNum
    elem.dataFld = "Article" + currFieldNum
}

function toggleComplete() {
    if (document.all.buttonWrapper.className == "") {
        document.all.display.dataSrc = "#rights_raw"
        document.all.display.dataFld = "column1"
        document.all.display.dataFormatAs = "text"
        document.all.buttonWrapper.className = "hiddenControl"
    } else {
        document.all.display.dataSrc = "#rights_html"
        document.all.display.dataFld = "Article1"
        document.all.display.dataFormatAs = "HTML"
        document.all.buttonWrapper.className = ""
    }
}
</SCRIPT>
</HEAD>
<BODY>
<P><B>U.S. Bill of Rights</B></P>
<FORM>
<INPUT TYPE="button" VALUE="Toggle Complete/Individual"
onClick="toggleComplete()">
<SPAN ID="buttonWrapper" CLASS="">
<INPUT TYPE="button" VALUE="Prev" onClick="prevField()">
<INPUT TYPE="button" VALUE="Next" onClick="nextField()">
</SPAN>
</FORM>


<DIV ID="display" DATASRC="#rights_html" DATAFLD="Article1"
DATAFORMATAS="HTML"></DIV>

<OBJECT ID="rights_html" CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME="DataURL" VALUE="Bill of Rights.txt">
    <PARAM NAME="UseHeader" VALUE="True">
    <PARAM NAME="FieldDelim" VALUE="&#09;">
</OBJECT>
<OBJECT ID="rights_raw" CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME="DataURL" VALUE="Bill of Rights (no format).txt">
    <PARAM NAME="FieldDelim" VALUE="\">
    <PARAM NAME="RowDelim" VALUE="\">
</OBJECT>
</BODY>
</HTML>
```

Another button on the page enables you to switch between the initial piecemeal version of the document and the unformatted version in its entirety. To load the entire document as a single record, the `FieldDelim` and `RowDelim` parameters of the second OBJECT element eliminate their default values by replacing them with characters that don't appear in the document at all. And because the external file does not have a field name in the file, the default value (`column1` for the lone column in this document) is the data field. Thus, in the `toggleComplete()` function, the `dataSrc` property is changed to the desired OBJECT element ID, the `dataFld` property is set to the correct value for the data source, and the `dataFormatAs` property is changed to reflect the different intention of the source content (to be rendered as HTML or as plain text). When the display shows the entire document, you can hide the two radio buttons by assigning a `className` value to the SPAN element that surrounds the buttons. The `className` value is the identifier of the class selector in the document's style sheet. When the `toggleComplete()` function resets the `className` property to empty, the default properties (normal inline display style) take hold.

One further example demonstrates the kind of power available to the TDC under script control. Listing 15-9 displays table data from a tab-delimited file of Academy Award information. The data file has eight columns of data, and each column heading is treated as a field name: Year, Best Picture, Best Director, Best Director Film, Best Actress, Best Actress Film, Best Actor, and Best Actor Film. For the design of the page, only five fields from each record appear: Year, Film, Director, Actress, and Actor. Notice in the listing how the HTML for the table and its content is bound to the data source object and the fields within the data.

The "dynamic" part of this example is apparent in how you can sort and filter the data, once loaded into the browser, without further access to the original source data. The TDC object features `Sort` and `Filter` properties that enable you to act on the data currently loaded in the browser. The simplest kind of sorting indicates on which field (or fields, via a semicolon delimited list of field names) the entire data set should be sorted. Leading the name of the sort field is either a plus (to indicate ascending) or minus (descending) symbol. After setting the `data` object's `Sort` property, invoke its `Reset()` method to tell the object to apply the new property. The data in the bound table is immediately redrawn to reflect any changes.

Similarly, you can tell a data collection to display records that meet specific criteria. In Listing 15-9, two select lists and a pair of radio buttons provide the interface to the `Filter` property's settings (see Figure 1-1). For example, you can filter the output to display only those records in which the Best Picture was the same picture of the winning Best Actress's performance. Simple filter expressions are based on field names:

```
dataObj.Filter = "Best Picture" = "Best Actress Film"
```

Listing 15-9: **Sorting and Filtering Bound Data**

```
<HTML>
<HEAD>
<TITLE>Data Binding—Sorting</TITLE>
<SCRIPT LANGUAGE="JavaScript">
```

```
function sortByYear(type) {
    oscars.Sort = (type == "normal") ? "-Year" : "+Year"
    oscars.Reset()
}
function filterInCommon(form) {
    var filterExpr1 = form.filter1.options[form.filter1.selectedIndex].value
    var filterExpr2 = form.filter2.options[form.filter2.selectedIndex].value
    var operator = (form.operator[0].checked) ? "=" : "<>"
    var filterExpr = filterExpr1 + operator + filterExpr2
    oscars.Filter = filterExpr
    oscars.Reset()
}
</SCRIPT>

</HEAD>
<BODY>
<P><B>Academy Awards 1978-1997</B></P>
<FORM>
<P>Sort list by year <A HREF="javascript:sortByYear('normal')">from newest to
oldest</A> or <A HREF="javascript:sortByYear('reverse')">from oldest to
newest</A>.</P>
<P>Filter listings for records whose
<SELECT NAME="filter1" onChange="filterInCommon(this.form)">
    <OPTION VALUE="">
    <OPTION VALUE="Best Picture">Best Picture
    <OPTION VALUE="Best Director Film">Best Director's Film
    <OPTION VALUE="Best Actress Film">Best Actress's Film
    <OPTION VALUE="Best Actor Film">Best Actor's Film
</SELECT>
<INPUT TYPE="radio" NAME="operator" CHECKED
onClick="filterInCommon(this.form)">is
<INPUT TYPE="radio" NAME="operator" onClick="filterInCommon(this.form)">is not
<SELECT NAME="filter2" onChange="filterInCommon(this.form)">
    <OPTION VALUE="">
    <OPTION VALUE="Best Picture">Best Picture
    <OPTION VALUE="Best Director Film">Best Director's Film
    <OPTION VALUE="Best Actress Film">Best Actress's Film
    <OPTION VALUE="Best Actor Film">Best Actor's Film
</SELECT>
</P>
</FORM>
<TABLE DATASRC="#oscars" BORDER=1 ALIGN="center">
<THEAD STYLE="background-color:yellow; text-align:center">
<TR><TD>Year</TD>
    <TD>Film</TD>
    <TD>Director</TD>
    <TD>Actress</TD>
    <TD>Actor</TD>
</TR>
</THEAD>
<TR>
```

Continued

Listing 15-9 *(continued)*

```
    <TD><DIV ID="col1" DATAFLD="Year" ></DIV></TD>
    <TD><DIV ID="col2" DATAFLD="Best Picture"></DIV></TD>
    <TD><DIV ID="col3" DATAFLD="Best Director"></DIV></TD>
    <TD><DIV ID="col4" DATAFLD="Best Actress"></DIV></TD>
    <TD><DIV ID="col5" DATAFLD="Best Actor"></DIV></TD>
</TR>
</TABLE>

<OBJECT ID="oscars" CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME="DataURL" VALUE="Academy Awards.txt">
    <PARAM NAME="UseHeader" VALUE="True">
    <PARAM NAME="FieldDelim" VALUE="&#09;">
</OBJECT>
</BODY>
</HTML>
```

For more detailed information on Data Source Objects and their properties, visit `http://msdn.microsoft.com` and search for "Data Binding."

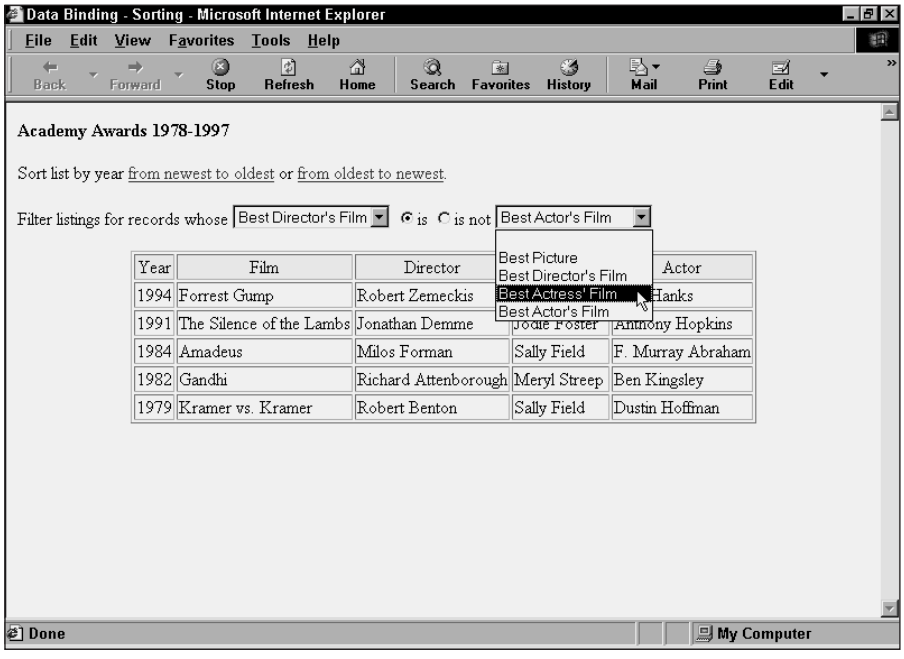


Figure 1-1: IE/Windows data binding puts filtering, sorting, and display under script control.

dir

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Changing this property value in a standard U.S. version of the browser only makes the right margin the starting point for each new line of text (in other words, the characters are not rendered in reverse order). You can experiment with this in The Evaluator by entering the following statements into the expression evaluation field:

```
document.getElementById("myP").dir = "rtl"
```

disabled

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				(✓)			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `disabled` property on both form elements (IE4+ and NN6) and regular HTML elements (IE5.5). For IE4+ and NN6, see what happens when you disable the output textarea by entering the following statement into the top text box:

```
document.forms[0].output.disabled = true
```

The textarea is disabled for user entry, although you can still set the field's `value` property via script (which is how the `true` returned value got there).

If you have IE5.5+, disable the `myP` element by entering the following statement into the top text box:

```
document.all.myP.disabled = true
```

The sample paragraph's text turns gray.

document

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The following simplified function accepts a parameter that can be any object in a document hierarchy. The script finds out the reference of the object's containing document for further reference to other objects:

```
function getCompanionFormCount(obj) {
    var ownerDoc = obj.document
    return ownerDoc.forms.length
}
```

Because the `ownerDoc` variable contains a valid reference to a `document` object, the `return` statement uses that reference to return a typical property of the document object hierarchy.

firstChild
lastChild

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

These two properties come in handy for Listing 15-10, whose job it is to either add or replace `LI` elements to an existing `OL` element. You can enter any text you want to appear at the beginning or end of the list. Using the `firstChild` and `lastChild` properties simplifies access to the ends of the list. For the functions that replace child nodes, the example uses the `replaceChild()` method. Alternatively for IE4+, you can modify the `innerText` property of the objects returned by the `firstChild` or `lastChild` property. This example is especially interesting to watch when you add items to the list: The browser automatically renumbers items to fit the current state of the list.

Listing 15-10: **Using firstChild and lastChild Properties**

```
<HTML>
<HEAD>
<TITLE>firstChild and lastChild Properties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// helper function for prepend() and append()
function makeNewLI(txt) {
    var newItem = document.createElement("LI")
    newItem.innerHTML = txt
    return newItem
}
function prepend(form) {
    var newItem = makeNewLI(form.input.value)
    var firstLI = document.getElementById("myList").firstChild
    document.getElementById("myList").insertBefore(newItem, firstLI)
```

```
}
function append(form) {
    var newItem = makeNewLI(form.input.value)
    var lastLI = document.getElementById("myList").lastChild
    document.getElementById("myList").appendChild(newItem)
}
function replaceFirst(form) {
    var newItem = makeNewLI(form.input.value)
    var firstLI = document.getElementById("myList").firstChild
    document.getElementById("myList").replaceChild(newItem, firstLI)
}
function replaceLast(form) {
    var newItem = makeNewLI(form.input.value)
    var lastLI = document.getElementById("myList").lastChild
    document.getElementById("myList").replaceChild(newItem, lastLI)
}
</SCRIPT>

</HEAD>
<BODY>
<H1>firstChild and lastChild Property Lab</H1>
<HR>
<FORM>
<LABEL>Enter some text to add to or replace in the OL element:</LABEL><BR>
<INPUT TYPE="text" NAME="input" SIZE=50><BR>
<INPUT TYPE="button" VALUE="Insert at Top" onClick="prepend(this.form)">
<INPUT TYPE="button" VALUE="Append to Bottom" onClick="append(this.form)">
<BR>
<INPUT TYPE="button" VALUE="Replace First Item"
onClick="replaceFirst(this.form)">
<INPUT TYPE="button" VALUE="Replace Last Item" onClick="replaceLast(this.form)">
</FORM>
<P></P>
<OL ID="myList"><LI>Initial Item 1
<LI>Initial Item 2
<LI>Initial Item 3
<LI>Initial Item 4
<OL>
</BODY>
</HTML>
```

height
width

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

The following example increases the width of a table by 10 percent.

```
var tableW = parseInt(document.all.myTable.width)
document.all.myTable.width = (tableW * 1.1) + "%"
```

Because the initial setting for the `WIDTH` attribute of the TABLE element is set as a percentage value, the script calculation extracts the number from the percentage width string value. In the second statement, the old number is increased by 10 percent and turned into a percentage string by appending the percentage symbol to the value. The resulting string value is assigned to the `width` property of the table.

hideFocus

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `hideFocus` property in IE5.5. Enter the following statement into the top text field to assign a `tabIndex` value to the `myP` element so that, by default, the element receives focus and the dotted rectangle:

```
document.all.myP.tabIndex = 1
```

Press the Tab key several times until the paragraph receives focus. Now, disable the focus rectangle:

```
document.all.myP.hideFocus = true
```

If you now press the Tab key several times, the dotted rectangle does not appear around the paragraph. To prove that the element still receives focus, scroll the page down to the bottom so that the paragraph is not visible (you may have to resize the window). Click one of the focusable elements at the bottom of the page, and then press the Tab key slowly until the Address field toolbar has focus. Press the Tab key once. The page scrolls to bring the paragraph into view, but there is no focus rectangle around the element.

id

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Rarely do you need to access this property in a script — unless you write an authoring tool that iterates through all elements of a page to extract the IDs assigned by the author. You can retrieve an object reference once you know the object's `id` property (via the `document.getElementById(elemID)` method). But if for some reason your script doesn't know the ID of, say, the second paragraph of a document, you can extract that ID as follows:

```
var elemID = document.all.tags("P")[1].id
```

innerHTML
innerText

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				(✓)			✓	✓	✓

Example

The IE4+ page generated by Listing 15-11 contains an `H1` element label and a paragraph of text. The purpose is to demonstrate how the `innerHTML` and `innerText` properties differ in their intent. Two text boxes contain the same combination of text and HTML tags that replaces the inner content of the paragraph's label.

If you apply the default content of the first text box to the `innerHTML` property of the `label1` object, the italic style is rendered as such for the first word. In addition, the text in parentheses is rendered with the help of the small style sheet rule assigned by virtue of the surrounding `<SPAN>` tags. But if you apply that same content to the `innerText` property of the `label` object, the tags are rendered as is.

Use this as a laboratory to experiment with some other content in both text boxes. See what happens when you insert a `<BR>` tag within some text of both text boxes.

Listing 15-11: **Using innerHTML and innerText Properties**

```
<HTML>
<HEAD>
<TITLE>innerHTML and innerText Properties</TITLE>
<STYLE TYPE="text/css">
H1 {font-size:18pt; font-weight:bold; font-family:"Comic Sans MS", Arial, sans-
serif}
.small {font-size:12pt; font-weight:400; color:gray}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">

function setGroupLabelAsText(form) {
    var content = form.textInput.value
```

Continued

Listing 15-11 *(continued)*

```
    if (content) {
        document.all.label1.innerText = content
    }
}
function setGroupLabelAsHTML(form) {
    var content = form.HTMLInput.value
    if (content) {
        document.all.label1.innerHTML = content
    }
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<P>
    <INPUT TYPE="text" NAME="HTMLInput"
    VALUE="<I>First</I> Article <SPAN CLASS='small'>(of ten)</SPAN>"
    SIZE=50>
    <INPUT TYPE="button" VALUE="Change Heading HTML"
    onClick="setGroupLabelAsHTML(this.form)">
</P>
<P>
    <INPUT TYPE="text" NAME="textInput"
    VALUE="<I>First</I> Article <SPAN CLASS='small'>(of ten)</SPAN>"
    SIZE=50>
    <INPUT TYPE="button" VALUE="Change Heading Text"
    onClick="setGroupLabelAsText(this.form)">
</P>
</FORM>
<H1 ID="label1">ARTICLE I</H1>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of speech, or of
the press; or the right of the people peaceably to assemble, and to petition the
government for a redress of grievances.
</P>
</BODY>
</HTML>
```

isContentEditable

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with both the `contentEditable` and `isContentEditable` properties on the `myP` and nested `myEM` elements (reload the page to start with a known version). Check the current setting for the `myEM` element by typing the following statement into the top text field:

```
myEM.isContentEditable
```

This value is `false` because no element upward in the element containment hierarchy is set to be editable yet. Next, turn on editing for the surrounding `myP` element:

```
myP.contentEditable = true
```

At this point, the entire `myP` element is editable because its child element is set, by default, to inherit the edit state of its parent. Prove it by entering the following statement into the top text box:

```
myEM.isContentEditable
```

While the `myEM` element is shown to be editable, no change has accrued to its `contentEditable` property:

```
myEM.contentEditable
```

This property value remains the default `inherit`. You can see an additional example of these two properties in use in Listing 15-7.

isDisabled

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with both the `disabled` and `isDisabled` properties on the `myP` and nested `myEM` elements (reload the page to start with a known version). Check the current setting for the `myEM` element by typing the following statement into the top text field:

```
myEM.isDisabled
```

This value is `false` because no element upward in the element containment hierarchy is set for disabling yet. Next, disable the surrounding `myP` element:

```
myP.disabled = true
```

At this point, the entire `myP` element (including its children) is disabled. Prove it by entering the following statement into the top text box:

```
myEM.isDisabled
```

While the `myEM` element is shown as disabled, no change has accrued to its `disabled` property:

```
myEM.disabled
```

This property value remains the default `false`.

isMultiLine

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to read the `isMultiLine` property for elements on that page. Try the following statements in the top text box:

```
document.body.isMultiLine
document.forms[0].input.isMultiLine
myP.isMultiLine
myEM.isMultiLine
```

All but the text field form control report that they are capable of occupying multiple lines.

isTextEdit

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Good coding practice dictates that your script check for this property before invoking the `createTextRange()` method on any object. A typical implementation is as follows:

```
if (document.all.myObject.isTextEdit) {
    var myRange = document.all.myObject.createTextRange()
    [more statements that act on myRange]
}
```

lang

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Values for the `lang` property consist of strings containing valid ISO language codes. Such codes have, at the minimum, a primary language code (for example, `"fr"` for French) plus an optional region specifier (for example, `"fr-ch"` for Swiss French). The code to assign a Swiss German value to an element looks like the following:

```
document.all.specialSpan.lang = "de-ch"
```

language

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Although it is unlikely that you will modify this property, the following example shows you how to do it for a table cell object:

```
document.all.cellA3.language = "vbs"
```

lastChild

See `firstchild`.

length

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

You can try the following sequence of statements in the top text box of The Evaluator (Chapter 13 in the *JavaScript Bible*) to see how the `length` property returns values (and sets them for some objects). Note that some statements work in only some browser versions.

```
(All browsers)  document.forms.length
(All browsers)  document.forms[0].elements.length
(NN3+, IE4+)    document.images.length
(NN4+)          document.layers.length
(IE4+)          document.all.length
(IE5+, NN6)     document.getElementById("myTable").childNodes.length
```

nextSibling
previousSibling

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

The following function assigns the same class name to all child nodes of an element:

```
function setAllChildClasses(parentElem, className) {
    var childElem = parentElem.firstChild
    while (childElem.nextSibling) {
        childElem.className = className
        childElem = childElem.nextSibling
    }
}
```

This example is certainly not the only way to achieve the same results. Using a `for` loop to iterate through the `childNodes` collection of the parent element is an equally valid approach.

nodeName

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

The following function demonstrates one (not very efficient) way to assign a new class name to every P element in an IE5+ document:

```
function setAllPClasses(className) {
    for (var i = 0; i < document.all.length; i++) {
        if (document.all[i].nodeName == "P") {
            document.all[i].className = className
        }
    }
}
```

A more efficient approach uses the `getElementsByTagName()` method to retrieve a collection of all P elements and then iterate through them directly.

nodeType

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

You can experiment with viewing `nodeType` property values in The Evaluator. The P element whose ID is `myP` is a good place to start. The P element itself is a `nodeType` of `1`:

```
document.getElementById("myP").nodeType
```

This element has three child nodes: a string of text (`nodeName #text`); an EM element (`nodeName EM`); and the rest of the text of the element content (`nodeName #text`). If you view the `nodeType` of either of the text portions, the value comes back as `3`:

```
document.getElementById("myP").childNodes[0].nodeType
```

In NN6 and IE5/Mac, you can inspect the `nodeType` of the one attribute of this element (the `ID` attribute):

```
document.getElementById("myP").attributes[0].nodeType
```

With NN6 and IE5/Mac, you can see how the `document` object returns a `nodeType` of `9`:

```
document.nodeType
```

When IE5 does not support a `nodeType` constant for a node, its value is sometimes reported as `1`. However, more likely the value is `undefined`.

nodeValue

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

The first example increases the width of a TEXTAREA object by 10 percent. The `nodeValue` is converted to an integer (for NN6's string values) before performing the math and reassignment:

```
function widenCols(textareaElem) {
    var colWidth = parseInt(textareaElem.attributes["cols"].nodeValue, 10)
    textareaElem.attributes["cols"].nodeValue = (colWidth * 1.1)
}
```

The second example replaces the text of an element, assuming that the element contains no further nested elements:

```
function replaceText(elem, newText) {
    if (elem.childNodes.length == 1 && elem.firstChild.nodeType == 3) {
        elem.firstChild.nodeValue = newText
    }
}
```

The function builds in one final verification that the element contains just one child node and that it is a text type. An alternative version of the assignment statement of the second example uses the `innerText` property in IE with identical results:

```
elem.innerText = newText
```

offsetHeight
offsetWidth

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

With IE4+, you can substitute the `offsetHeight` and `offsetWidth` properties for `clientHeight` and `clientWidth` in Listing 15-6. The reason is that the two elements in question have their widths hard-wired in style sheets. Thus, the `offsetWidth` property follows that lead rather than observing the default width of the parent (BODY) element.

With IE5+ and NN6, you can use The Evaluator to inspect the `offsetHeight` and `offsetWidth` property values of various objects on the page. Enter the following statements into the top text box:

```
document.getElementById("myP").offsetWidth
document.getElementById("myEM").offsetWidth
document.getElementById("myP").offsetHeight
document.getElementById("myTable").offsetWidth
```

offsetLeft
offsetTop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following IE script statements utilize all four "offset" dimensional properties to size and position a DIV element so that it completely covers a SPAN element located within a P element. This can be for a fill-in-the-blank quiz that provides text entry fields elsewhere on the page. As the user gets an answer correct, the blocking `DIV` element is hidden to reveal the correct answer.

```
document.all.blocker.style.pixelLeft = document.all.span2.offsetLeft
document.all.blocker.style.pixelTop = document.all.span2.offsetTop
document.all.blockImg.height = document.all.span2.offsetHeight
document.all.blockImg.width = document.all.span2.offsetWidth
```

Because the `offsetParent` property for the SPAN element is the BODY element, the positioned DIV element can use the same positioning context (it's the default context, anyway) for setting the `pixelLeft` and `pixelTop` style properties. (Remember that positioning properties belong to an element's `style` object.) The `offsetHeight` and `offsetWidth` properties can read the dimensions of the SPAN element (the example has no borders, margins, or padding to worry about) and assign them to the dimensions of the image contained by the blocker `DIV` element.

This example is also a bit hazardous in some implementations. If the text of `span2` wraps to a new line, the new `offsetHeight` value has enough pixels to accommodate both lines. But the `blockImg` and `blocker` DIV elements are block-level elements that render as a simple rectangle. In other words, the `blocker` element doesn't turn into two separate strips to cover the pieces of `span2` that spread across two lines.

offsetParent

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

You can use the `offsetParent` property to help you locate the position of a nested element on the page. Listing 15-12 demonstrates how a script can "walk" up the hierarchy of `offsetParent` objects in IE for Windows to assemble the location of a nested element on a page. The goal of the exercise in Listing 15-12 is to position an image at the upper-left corner of the second table cell. The entire table is centered on the page.

The `onLoad` event handler invokes the `setImagePosition()` function. The function first sets a Boolean flag that determines whether the calculations should be based on the client or offset sets of properties. IE4/Windows and IE5/Mac rely on client properties, while IE5+/Windows works with the offset properties. The discrepancies even out, however, with the `while` loop. This loop traverses the `offsetParent` hierarchy starting with the `offsetParent` of the cell out to, but not including, the `document.body` object. The `body` object is not included because that is the positioning context for the image. In IE5, the `while` loop executes only once

because just the TABLE element exists between the cell and the body; in IE4, the loop executes twice to account for the TR and TABLE elements up the hierarchy. Finally, the cumulative values of left and top measures are applied to the positioning properties of the DIV object's style and the image is made visible.

Listing 15-12: **Using the offsetParent Property**

```
<HTML>
<HEAD>
<TITLE>offsetParent Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setImagePosition(){
    var cElement = document.all.myCell
    // Set flag for whether calculations should use
    // client- or offset- property measures. Use
    // client- for IE5/Mac and IE4/Windows; otherwise
    // use offset- properties. An ugly, but necessary
    // workaround.
    var useClient = (cElement.offsetTop == 0) ?
        ((cElement.offsetParent.tagName == "TR") ? false : true) : false
    if (useClient) {
        var x = cElement.clientLeft
        var y = cElement.clientTop
    } else {
        var x = cElement.offsetLeft
        var y = cElement.offsetTop
    }
    var pElement = document.all.myCell.offsetParent
    while (pElement != document.body) {
        if (useClient) {
            x += pElement.clientLeft
            y += pElement.clientTop
        } else {
            x += pElement.offsetLeft
            y += pElement.offsetTop
        }
        pElement = pElement.offsetParent
    }
    document.all.myDIV.style.pixelLeft = x
    document.all.myDIV.style.pixelTop = y
    document.all.myDIV.style.visibility = "visible"
}
</SCRIPT>
</HEAD>
<BODY onload="setImagePosition()">
<SCRIPT LANGUAGE="JavaScript">
</SCRIPT>
<H1>The offsetParent Property</H1>
<HR>
<P>After the document loads, the script positions a small image in the upper
left corner of the second table cell.</P>
<TABLE BORDER=1 ALIGN="center">
```

```
<TR>
    <TD>This is the first cell</TD>
    <TD ID="myCell">This is the second cell.</TD>
</TR>
</TABLE>
<DIV ID="myDIV" STYLE="position:absolute; visibility:hidden; height:12;
width:12">
<IMG SRC="end.gif" HEIGHT=12 WIDTH=12></DIV>
</BODY>
</HTML>
```

outerHTML
outerText

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The page generated by Listing 15-13 (IE4+/Windows only) contains an `H1` element label and a paragraph of text. The purpose is to demonstrate how the `outerHTML` and `outerText` properties differ in their intent. Two text boxes contain the same combination of text and HTML tags that replaces the element that creates the paragraph's label.

If you apply the default content of the first text box to the `outerHTML` property of the `label1` object, the H1 element is replaced by a SPAN element whose `CLASS` attribute acquires a different style sheet rule defined earlier in the document. Notice that the ID of the new SPAN element is the same as the original H1 element. This allows the script attached to the second button to address the object. But this second script replaces the element with the raw text (including tags). The element is now gone, and any attempt to change the `outerHTML` or `outerText` properties of the `label1` object causes an error because there is no longer a `label1` object in the document.

Use this laboratory to experiment with some other content in both text boxes.

Listing 15-13: **Using outerHTML and outerText Properties**

```
<HTML>
<HEAD>
<TITLE>outerHTML and outerText Properties</TITLE>
<STYLE TYPE="text/css">
```

Continued

Listing 15-13 *(continued)*

```
H1 {font-size:18pt; font-weight:bold; font-family:"Comic Sans MS", Arial, sans-
serif}
.heading {font-size:20pt; font-weight:bold; font-family:"Arial Black", Arial,
sans-serif}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">

function setGroupLabelAsText(form) {
    var content = form.textInput.value
    if (content) {
        document.all.label1.outerText = content
    }
}
function setGroupLabelAsHTML(form) {
    var content = form.HTMLInput.value
    if (content) {
        document.all.label1.outerHTML = content
    }
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<P>
    <INPUT TYPE="text" NAME="HTMLInput"
    VALUE="<SPAN ID='label1' CLASS='heading'>Article the First</SPAN>" SIZE=55>
    <INPUT TYPE="button" VALUE="Change Heading HTML"
    onClick="setGroupLabelAsHTML(this.form)">
</P>
<P>
    <INPUT TYPE="text" NAME="textInput"
    VALUE="<SPAN ID='label1' CLASS='heading'>Article the First</SPAN>" SIZE=55>
    <INPUT TYPE="button" VALUE="Change Heading Text"
    onClick="setGroupLabelAsText(this.form)">
</P>
</FORM>
<H1 ID="label1">ARTICLE I</H1>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of speech, or of
the press; or the right of the people peaceably to assemble, and to petition the
government for a redress of grievances.
</P>
</BODY>
</HTML>
```

ownerDocument

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to explore the `ownerDocument` property in NN6. Enter the following statement into the top text box:

```
document.body.childNodes[5].ownerDocument
```

The result is a reference to the `document` object. You can use that to inspect a property of the document, as shown in the following statement you should enter into the top text box:

```
document.body.childNodes[5].ownerDocument.URL
```

This returns the `document.URL` property for the document that owns the child node.

parentElement

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

You can experiment with the `parentElement` property in The Evaluator. The document contains a P element named myP. Type each of the following statements from the left column into the upper expression evaluation text box and press Enter to see the results.

Expression	*Result*
`document.all.myP.tagName`	`P`
`document.all.myP.parentElement`	`[object]`
`document.all.myP.parentElement.tagName`	`BODY`
`document.all.myP.parentElement.parentElement`	`[object]`
`document.all.myP.parentElement.parentElement.tagName`	`HTML`
`document.all.myP.parentElement.parentElement.parentElement`	`null`

parentNode

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Use The Evaluator to examine the `parentNode` property values of both an element and a non-element node. Begin with the following two statements and watch the results of each:

```
document.getElementById("myP").parentNode.tagName
document.getElementById("myP").parentElement.tagName    (IE only)
```

Now examine the properties from the point of view of the first text fragment node of the `myP` paragraph element:

```
document.getElementById("myP").childNodes[0].nodeValue
document.getElementById("myP").childNodes[0].parentNode.tagName
document.getElementById("myP").childNodes[0].parentElement    (IE only)
```

Notice (in IE) that the text node does not have a `parentElement` property.

parentTextEdit

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The page resulting from Listing 15-14 contains a paragraph of Greek text and three radio buttons that select the size of a paragraph chunk: one character, one word, or one sentence. If you click anywhere within the large paragraph, the `onClick` event handler invokes the `selectChunk()` function. The function first examines which of the radio buttons is selected to determine how much of the paragraph to highlight (select) around the point at which the user clicks.

After the script employs the `parentTextEdit` property to test whether the clicked element has a valid parent capable of creating a text range, it calls upon the property again to help create the text range. From there, `TextRange` object methods shrink the range to a single insertion point, move that point to the spot nearest the cursor location at click time, expand the selection to encompass the desired chunk, and select that bit of text.

Notice one workaround for the `TextRange` object's `expand()` method anomaly: If you specify a sentence, IE doesn't treat the beginning of a P element as the starting end of a sentence automatically. A camouflaged (white text color) period is appended to the end of the previous element to force the `TextRange` object to expand only to the beginning of the first sentence of the targeted P element.

Listing 15-14: **Using the parentTextEdit Property**

```
<HTML>
<HEAD>
<TITLE>parentTextEdit Property</TITLE>
<STYLE TYPE="text/css">
P {cursor:hand}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function selectChunk() {
    var chunk, range
    for (var i = 0; i < document.forms[0].chunk.length; i++) {
        if (document.forms[0].chunk[i].checked) {
            chunk = document.forms[0].chunk[i].value
            break
        }
    }
    var x = window.event.clientX
    var y = window.event.clientY
    if (window.event.srcElement.parentTextEdit) {
        range = window.event.srcElement.parentTextEdit.createTextRange()
        range.collapse()
        range.moveToPoint(x, y)
        range.expand(chunk)
        range.select()
    }
}
</SCRIPT>
</HEAD>

<BODY BGCOLOR="white">
<FORM>
<P>Choose how much of the paragraph is to be selected when you click anywhere in
it:<BR>
<INPUT TYPE="radio" NAME="chunk" VALUE="character" CHECKED>Character
<INPUT TYPE="radio" NAME="chunk" VALUE="word">Word
<INPUT TYPE="radio" NAME="chunk" VALUE="sentence">Sentence
<FONT COLOR="white">.</FONT></P>
</FORM>

<P onClick="selectChunk()">
Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim adminim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit involuptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.
</P>
</BODY>
</HTML>
```

previousSibling

See nextSibling.

readyState

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

To witness a readyState property other than complete for standard HTML, you can try examining the property in a script that immediately follows an <IMG> tag:

```
...
<IMG ID="myImg" SRC="someImage.gif">
<SCRIPT LANGUAGE="JavaScript">
alert(document.all.myImg.readyState)
</SCRIPT>
...
```

Putting this fragment into a document that is accessible across a slow network helps. If the image is not in the browser's cache, you might get the uninitialized or loading result. The former means that the IMG object exists, but it has not started receiving the image data from the server yet. If you reload the page, chances are that the image will load instantaneously from the cache and the readyState property will report complete.

recordNumber

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

You can see the recordNumber property in action in Listing 15-15. The data source is a small, tab-delimited file consisting of 20 records of Academy Award data. Thus, the table that displays a subset of the fields is bound to the data source object. Also bound to the data source object are three SPAN objects embedded within a paragraph near the top of the page. As the user clicks a row of data, three fields from that clicked record are placed into the bound SPAN objects.

The script part of this page is a mere single statement. When the user triggers the onClick event handler of the repeated TR object, the function receives a reference to the TR object as a parameter. The data store object maintains an internal copy of the data in a recordset object. One of the properties of this recordset object is

the `AbsolutePosition` property, which is the integer value of the current record that the data object points to (it can point to only one row at a time, and the default row is the first row). The statement sets the `AbsolutePosition` property of the `recordset` object to the `recordNumber` property for the row that the user clicks. Because the three SPAN elements are bound to the same data source, they are immediately updated to reflect the change to the data object's internal pointer to the current record. Notice, too, that the third SPAN object is bound to one of the data source fields not shown in the table. You can reach any field of a record because the Data Source Object holds the entire data source content.

Listing 15-15: **Using the Data Binding recordNumber Property**

```
<HTML>
<HEAD>
<TITLE>Data Binding (recordNumber)</TITLE>
<STYLE TYPE="text/css">
.filmTitle {font-style:italic}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
// set recordset pointer to the record clicked on in the table.
function setRecNum(row) {
    document.oscars.recordset.AbsolutePosition = row.recordNumber
}
</SCRIPT>

</HEAD>
<BODY>
<P><B>Academy Awards 1978-1997</B> (Click on a table row to extract data from
one record.)</P>
<P>The award for Best Actor of <SPAN DATASRC="#oscars" DATAFLD="Year"></SPAN>
 went to <SPAN DATASRC="#oscars" DATAFLD="Best Actor"></SPAN>
 for his outstanding achievement in the film
<SPAN CLASS="filmTitle" DATASRC="#oscars" DATAFLD="Best Actor Film"></SPAN>.</P>
<TABLE BORDER=1 DATASRC="#oscars" ALIGN="center">
<THEAD STYLE="background-color:yellow; text-align:center">
<TR><TD>Year</TD>
    <TD>Film</TD>
    <TD>Director</TD>
    <TD>Actress</TD>
    <TD>Actor</TD>
</TR>
</THEAD>
<TR ID=repeatableRow onClick="setRecNum(this)">
    <TD><DIV ID="col1" DATAFLD="Year"></DIV></TD>
    <TD><DIV CLASS="filmTitle" ID="col2" DATAFLD="Best Picture"></DIV></TD>
    <TD><DIV ID="col3" DATAFLD="Best Director"></DIV></TD>
    <TD><DIV ID="col4" DATAFLD="Best Actress"></DIV></TD>
    <TD><DIV ID="col5" DATAFLD="Best Actor"></DIV></TD>
```

Continued

Listing 15-15 *(continued)*

```
</TR>
</TABLE>

<OBJECT ID="oscars" CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME="DataURL" VALUE="Academy Awards.txt">
    <PARAM NAME="UseHeader" VALUE="True">
    <PARAM NAME="FieldDelim" VALUE="&#09;">
</OBJECT>
</BODY>
</HTML>
```

runtimeStyle

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to compare the properties of the `runtimeStyle` and `style` objects of an element. For example, an unmodified copy of The Evaluator contains an EM element whose ID is "myEM". Enter both

```
document.all.myEM.style.color
```

and

```
document.all.myEM.runtimeStyle.color
```

into the top text field in turn. Initially, both values are empty. Now assign a color to the `style` property via the upper text box:

```
document.all.myEM.style.color = "red"
```

If you now type the two earlier statements into the upper box, you can see that the `style` object reflects the change, while the `runtimeStyle` object still holds onto its original (empty) value.

scopeName

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

If you have a sample document that contains XML and a namespace spec, you can use `document.write()` or `alert()` methods to view the value of the `scopeName` property. The syntax is

```
document.all.elementID.scopeName
```

scrollHeight
scrollWidth

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with these two properties of the TEXTAREA object, which displays the output of evaluations and property listings. To begin, enter the following into the bottom one-line text field to list the properties of the `body` object:

```
document.body
```

This displays a long list of properties for the `body` object. Now enter the following property expression in the top one-line text field to see the `scrollHeight` property of the output TEXTAREA when it holds the dozens of lines of property listings:

```
document.all.output.scrollHeight
```

The result, some number probably in the hundreds, is now displayed in the output TEXTAREA. This means that you can scroll the content of the `output` element vertically to reveal that number of pixels. Click the Evaluate button once more. The result, 13 or 14, is a measure of the `scrollHeight` property of the TEXTAREA that had only the previous result in it. The scrollable height of that content was only 13 or 14 pixels, the height of the font in the TEXTAREA. The `scrollWidth` property of the output TEXTAREA is fixed by the width assigned to the element's `COLS` attribute (as calculated by the browser to determine how wide to make the textarea on the page).

scrollLeft
scrollTop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with these two properties of the TEXTAREA object, which displays the output of evaluations and property listings. To begin, enter the following into the bottom one-line text field to list the properties of the `body` object:

```
document.body
```

This displays a long list of properties for the `body` object. Use the TEXTAREA's scrollbar to page down a couple of times. Now enter the following property expression in the top one-line text field to see the `scrollTop` property of the output TEXTAREA after you scroll:

```
document.all.output.scrollTop
```

The result, some number, is now displayed in the output TEXTAREA. This means that the content of the `output` element was scrolled vertically. Click the Evaluate button once more. The result, `0`, is a measure of the `scrollTop` property of the TEXTAREA that had only the previous result in it. There wasn't enough content in the TEXTAREA to scroll, so the content was not scrolled at all. The `scrollTop` property, therefore, is zero. The `scrollLeft` property of the output is always zero because the TEXTAREA element is set to wrap any text that overflows the width of the element. No horizontal scrollbar appears in this case, and the `scrollLeft` property never changes.

sourceIndex

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

While the operation of this property is straightforward, the sequence of elements exposed by the `document.all` property may not be. To that end, you can use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment in IE4+ with the values that the `sourceIndex` property returns to see how the index values of the `document.all` collection follow the source code.

To begin, reload The Evaluator. Enter the following statement in the top text box to set a preinitialized global variable:

```
a = 0
```

When you evaluate this expression, a zero should appear in the Results box. Next, enter the following statement into the top text box:

```
document.all[a].tagName + " [" + a++ + "]"
```

There are a lot of plus signs in this statement, so be sure you enter it correctly. As you successively evaluate this statement (by repeatedly clicking the Evaluate button), the global variable (`a`) is incremented, thus enabling you to "walk through" the

elements in source code order. The `sourceIndex` value for each HTML tag appears in square brackets in the Results box. You generally begin with the following sequence:

```
HTML [0]
HEAD [1]
TITLE [2]
```

You can continue until there are no more elements, at which point an error message appears because the value of `a` exceeds the number of elements in the `document.all` array. Compare your findings against the HTML source code view of The Evaluator.

style

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Most of the action with the `style` property has to do with the `style` object's properties, so you can use The Evaluator here to simply explore the lists of `style` object properties available on as many DHTML-compatible browsers as you have running. To begin, enter the following statement into the lower, one-line text box to inspect the `style` property for the `document.body` object:

```
document.body.style
```

Now inspect the `style` property of the table element that is part of the original version of The Evaluator. Enter the following statement into the lower text box:

```
document.getElementById("myTable").style
```

In both cases, the values assigned to the `style` object's properties are quite limited by default.

tabIndex

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The HTML and scripting in Listing 15-16 demonstrate not only the way you can modify the tabbing behavior of a form on the fly, but also how to force form elements out of the tabbing sequence entirely in IE. In this page, the upper form (named `lab`) contains four elements. Scripts invoked by buttons in the lower form control the tabbing sequence. Notice that the `TABINDEX` attributes of all lower form elements are set to `-1`, which means that these control buttons are not part of the tabbing sequence in IE.

When you load the page, the default tabbing order for the `lab` form control elements (default setting of zero) takes charge. If you start pressing the Tab key, the precise results at first depend on the browser you use. In IE, the Address field is first selected; next the Tab sequence gives focus to the window (or frame, if this page were in a frameset); finally the tabbing reaches the `lab` form. Continue pressing the Tab key and watch how the browser assigns focus to each of the element types. In NN6, however, you must click anywhere on the content to get the Tab key to start working on form controls.

The sample script inverts the tabbing sequence with the help of a `for` loop that initializes two variables that work in opposite directions as the looping progresses. This gives the last element the lowest `tabIndex` value. The `skip2()` function simply sets the `tabIndex` property of the second text box to `-1`, removing it from the tabbing entirely (IE only). Notice, however, that you can click in the field and still enter text. (See the `disabled` property earlier in this chapter to see how to prevent field editing.) NN6 does not provide a `tabIndex` property setting that forces the browser to skip over a form control. You should disable the control instead.

Listing 15-16: **Controlling the tabIndex Property**

```
<HTML>
<HEAD>
<TITLE>tabIndex Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function invert() {
    var form = document.lab
    for (var i = 0, j = form.elements.length; i < form.elements.length;
    i++, j--) {
        form.elements[i].tabIndex = j
    }
}

function skip2() {
    document.lab.text2.tabIndex = -1
}

function resetTab() {
    var form = document.lab
    for (var i = 0; i < form.elements.length; i++) {
        form.elements[i].tabIndex = 0
    }
}
</SCRIPT>
</HEAD>

<BODY>
<H1>tabIndex Property Lab</H1>
<HR>
<FORM NAME="lab">
Text box no. 1: <INPUT TYPE="text" NAME="text1"><BR>
Text box no. 2: <INPUT TYPE="text" NAME="text2"><BR>
<INPUT TYPE="button" VALUE="A Button"><BR>
```

```
<INPUT TYPE="checkbox">And a checkbox
</FORM>
<HR>
<FORM NAME="control">
<INPUT TYPE="button" VALUE="Invert Tabbing Order" TABINDEX=-1
onClick="invert()"><BR>
<INPUT TYPE="button" VALUE="Skip Text box no. 2 (IE Only)" TABINDEX=-1
onClick="skip2()"><BR>
<INPUT TYPE="button" VALUE="Reset to Normal Order" TABINDEX=-1
onClick="resetTab()">
</FORM>
</BODY>
</HTML>
```

The final function, `resetTab()`, sets the `tabIndex` property value to zero for all `lab` form elements. This restores the default order; but in IE5.5/Windows, you may experience buggy behavior that prevents you from tabbing to items after you reset them. Only the reloading of the page provides a complete restoration of default behavior.

tagName

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

You can see the `tagName` property in action for the example associated with the `sourceIndex` property discussed earlier. In that example, the `tagName` property is read from a sequence of objects in source code order.

tagUrn

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

If you have a sample document that contains XML and a Namespace spec, you can use `document.write()` or `alert()` methods to view the value of the `tagUrn` property. The syntax is

```
document.all.elementID.tagUrn
```

title

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

You can see how dynamic a tooltip is in Listing 15-17. A simple paragraph element has its `TITLE` attribute set to `"First Time!"`, which is what the tooltip displays if you roll the pointer atop the paragraph and pause after the page loads. But an `onMouseOver` event handler for that element increments a global variable counter in the script, and the `title` property of the paragraph object is modified with each mouseover action. The `count` value is made part of a string assigned to the `title` property. Notice that there is not a live connection between the `title` property and the variable; instead, the new value explicitly sets the `title` property.

Listing 15-17: **Controlling the title Property**

```
<HTML>
<HEAD>
<TITLE>title Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// global counting variable
var count = 0

function setToolTip(elem) {
    elem.title = "You have previously rolled atop this paragraph " +
        count + " time(s)."
}

function incrementCount(elem) {
    count++
    setToolTip(elem)
}
</SCRIPT>

</HEAD>
<BODY>
<H1>title Property Lab</H1>
<HR>
<P ID="myP" TITLE="First Time!" onMouseOver="incrementCount(this)">
Roll the mouse over this paragraph a few times.<BR>
Then pause atop it to view the tooltip.</P>
</BODY>
</HTML>
```

uniqueID

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-18 demonstrates the recommended syntax for obtaining and applying a browser-generated identifier for an object. After you enter some text into the text box and click the button, the `addRow()` function appends a row to the table. The left column displays the identifier generated via the table row object's `uniqueID` property. IE5+ generates identifiers in the format `"ms__id`*n*`"`, where *n* is an integer starting with zero for the current browser session. Because the `addRow()` function assigns `uniqueID` values to the row and the cells in each row, the integer for each row is three greater than the previous one. There is no guarantee that future generations of the browser will follow this format, so do not rely on the format or sequence in your scripts.

Listing 15-18: Using the uniqueID Property

```
<HTML>
<HEAD>
<TITLE>Inserting an IE5+/Windows Table Row</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function addRow(item1) {
    if (item1) {
        // assign long reference to shorter var name
        var theTable = document.all.myTable
        // append new row to the end of the table
        var newRow = theTable.insertRow(theTable.rows.length)
        // give the row its own ID
        newRow.id = newRow.uniqueID

        // declare cell variable
        var newCell

        // an inserted row has no cells, so insert the cells
        newCell = newRow.insertCell(0)
        // give this cell its own id
        newCell.id = newCell.uniqueID
        // display the row's id as the cell text
        newCell.innerText = newRow.id
        newCell.bgColor = "yellow"
        // reuse cell var for second cell insertion
        newCell = newRow.insertCell(1)
        newCell.id = newCell.uniqueID
```

Continued

Listing 15-18 *(continued)*

```
        newCell.innerText = item1
    }
}
</SCRIPT>
</HEAD>

<BODY>
<TABLE ID="myTable" BORDER=1>
<TR>
<TH>Row ID</TH>
<TH>Data</TH>
</TR>

<TR ID="firstDataRow">
<TD>firstDataRow
<TD>Fred
</TR>
<TR ID="secondDataRow">
<TD>secondDataRow
<TD>Jane
</TR>
</TABLE>
<HR>
<FORM>
Enter text to be added to the table:<BR>
<INPUT TYPE="text" NAME="input" SIZE=25><BR>
<INPUT TYPE='button' VALUE='Insert Row' onClick='addRow(this.form.input.value)'>
</FORM>
</BODY>
</HTML>
```

Methods

addBehavior("*URL*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-19a is the JavaScript code for an external component named `makeHot.htc`. Its purpose is to turn the `color` style property of an object to either a

default color ("red") or any other color that is passed to the component. For details on the syntax of the <PUBLIC> tags, see Chapter 48 of the *JavaScript Bible*. The code presented here helps you see how the page and scripts in Listing 15-19b work.

Listing 15-19a: **The makeHot.htc Behavior Component**

```
<PUBLIC:ATTACH EVENT="onmousedown" ONEVENT="makeHot()" />
<PUBLIC:ATTACH EVENT="onmouseup" ONEVENT="makeNormal()" />
<PUBLIC:PROPERTY NAME="hotColor" />
<PUBLIC:METHOD NAME="setHotColor" />
<SCRIPT LANGUAGE="JScript">
var oldColor
var hotColor = "red"

function setHotColor(color) {
    hotColor = color
}

function makeHot() {
    if (event.srcElement == element) {
        oldColor = style.color
        runtimeStyle.color = hotColor
    }
}

function makeNormal() {
    if (event.srcElement == element) {
        runtimeStyle.color = oldColor
    }
}
</SCRIPT>
```

The object to which the component is attached is a simple paragraph object, shown in Listing 15-19b. When the page loads, the behavior is not attached, so clicking the paragraph text has no effect.

When you turn on the behavior by invoking the `turnOn()` function, the `addBehavior()` method attaches the code of the `makeHot.htc` component to the `myP` object. At this point, the `myP` object has one more property, one more method, and two more event handlers that are written to be made public by the component's code. If you want the behavior to apply to more than one paragraph in the document, you have to invoke the `addBehavior()` method for each paragraph object.

After the behavior file is instructed to start loading, the `setInitialColor()` function is called to set the new color property of the paragraph to the user's choice from the SELECT list. But this can happen only if the component is fully loaded. Therefore, the function checks the `readyState` property of `myP` for completeness before invoking the component's function. If IE is still loading the component, the function is invoked again in 500 milliseconds.

As long as the behavior is loaded, you can change the color used to turn the paragraph "hot." The function first ensures that the component is loaded by checking that the object has the new color property. If it does, then (as a demonstration of how to expose and invoke a component method) the method of the component is invoked. You can also simply set the property value.

Listing 15-19b: **Using addBehavior() and removeBehavior()**

```
<HTML>
<HEAD>
<TITLE>addBehavior() and removeBehavior() Methods</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var myPBehaviorID

function turnOn() {
    myPBehaviorID = document.all.myP.addBehavior("makeHot.htc")
    setInitialColor()
}

function setInitialColor() {
    if (document.all.myP.readyState == "complete") {
        var select = document.forms[0].colorChoice
        var color = select.options[select.selectedIndex].value
        document.all.myP.setHotColor(color)
    } else {
        setTimeout("setInitialColor()", 500)
    }
}

function turnOff() {
    document.all.myP.removeBehavior(myPBehaviorID)
}

function setColor(select, color) {
    if (document.all.myP.hotColor) {
        document.all.myP.setHotColor(color)
    } else {
        alert("This feature is not available. Turn on the Behavior first.")
        select.selectedIndex = 0
    }
}
function showBehaviorCount() {
    var num = document.all.myP.behaviorUrns.length
    var msg = "The myP element has " + num + " behavior(s). "
    if (num > 0) {
        msg += "Name(s): \r\n"
        for (var i = 0; i < num; i++) {
            msg += document.all.myP.behaviorUrns[i] + "\r\n"
        }
    }
    alert(msg)
}
```

```
</SCRIPT>
</HEAD>
<BODY>
<H1>addBehavior() and removeBehavior() Method Lab</H1>
<HR>
<P ID="myP">This is a sample paragraph. After turning on the behavior,
it will turn your selected color when you mouse down anywhere in this
paragraph.</P>
<FORM>
<INPUT TYPE="button" VALUE="Switch On Behavior" onClick="turnOn()">
Choose a 'hot' color:
<SELECT NAME="colorChoice" onChange="setColor(this, this.value)">
<OPTION VALUE="red">red
<OPTION VALUE="blue">blue
<OPTION VALUE="cyan">cyan
</SELECT><BR>
<INPUT TYPE="button" VALUE="Switch Off Behavior" onClick="turnOff()">
<P><INPUT TYPE="button" VALUE="Count the URNs"
onClick="showBehaviorCount()"></P>
</BODY>
</HTML>
```

To turn off the behavior, the `removeBehavior()` method is invoked. Notice that the `removeBehavior()` method is associated with the `myP` object, and the parameter is the ID of the behavior added earlier. If you associate multiple behaviors with an object, you can remove one without disturbing the others because each has its own unique ID.

`addEventListener("eventType", listenerFunc, useCapture)`
`removeEventListener("eventType", listenerFunc, useCapture)`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Listing 15-20 provides a compact workbench to explore and experiment with the basic W3C DOM event model. When the page loads, no event listeners are registered with the browser (except for the control buttons, of course). But you can add an event listener for a `click` event in bubble and/or capture mode to the BODY element or the P element that surrounds the SPAN holding the line of text. If you add an event listener and click the text, you see a readout of the element processing the

event and information indicating whether the event phase is bubbling (3) or capture (1). With all event listeners engaged, notice the sequence of events being processed. Remove listeners one at a time to see the effect on event processing.

Listing 15-20 includes code for event capture that does not operate in NN6. Event capture facilities should work in a future version of the browser.

Listing 15-20: **W3C Event Lab**

```
<HTML>
<HEAD>
<TITLE>W3C Event Model Lab</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
// add event listeners
function addBubbleListener(elemID) {
    document.getElementById(elemID).addEventListener("click", reportEvent, false)
}
function addCaptureListener(elemID) {
    document.getElementById(elemID).addEventListener("click", reportEvent, true)
}
// remove event listeners
function removeBubbleListener(elemID)  {
    document.getElementById(elemID).removeEventListener("click", reportEvent, false)
}
function removeCaptureListener(elemID) {
    document.getElementById(elemID).removeEventListener("click", reportEvent, true)
}
// display details about any event heard
function reportEvent(evt) {
    if (evt.target.parentNode.id == "mySPAN") {
        var msg = "Event processed at " + evt.currentTarget.tagName +
            " element (event phase = " + evt.eventPhase + ").\n"
        document.controls.output.value += msg
    }
}
// clear the details textarea
function clearTextArea() {
    document.controls.output.value = ""
}
</SCRIPT>
</HEAD>
<BODY ID="myBODY">
<H1>W3C Event Model Lab</H1>
<HR>
<P ID="myP"><SPAN ID="mySPAN">This paragraph (a SPAN element nested inside a P
element) can be set to listen for "click" events.</SPAN></P>
<HR>
<TABLE CELLPADDING=5 BORDER=1>
```

```
<CAPTION STYLE="font-weight:bold">Control Panel</CAPTION>
<FORM NAME="controls">
<TR STYLE="background-color:#ffff99"><TD ROWSPAN=2>"Bubble"-type click listener:
    <TD><INPUT TYPE="button" VALUE="Add to BODY"
        onClick="addBubbleListener('myBODY')">
    <TD><INPUT TYPE="button" VALUE="Remove from BODY"
        onClick="removeBubbleListener('myBODY')">
</TR>
<TR STYLE="background-color:#ffff99">
    <TD><INPUT TYPE="button" VALUE="Add to P"
        onClick="addBubbleListener('myP')">
    <TD><INPUT TYPE="button" VALUE="Remove from P"
        onClick="removeBubbleListener('myP')">
</TR>
<TR STYLE="background-color:#ff9999"><TD ROWSPAN=2>"Capture"-type click
listener:
    <TD><INPUT TYPE="button" VALUE="Add to BODY"
        onClick="addCaptureListener('myBODY')">
    <TD><INPUT TYPE="button" VALUE="Remove from BODY"
        onClick="removeCaptureListener('myBODY')">
</TR>
<TR STYLE="background-color:#ff9999">
    <TD><INPUT TYPE="button" VALUE="Add to P"
        onClick="addCaptureListener('myP')">
    <TD><INPUT TYPE="button" VALUE="Remove from P"
        onClick="removeCaptureListener('myP')">
</TR>
<P>Examine click event characteristics: <INPUT TYPE="button" VALUE="Clear"
onClick="clearTextArea()"><BR>
<TEXTAREA NAME="output" COLS="80" ROWS="6" WRAP="virtual"></TEXTAREA>
</FORM>
</TABLE>
</BODY>
</HTML>
```

appendChild(*elementObject*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Scripts in Listing 15-21 demonstrate how the three major child-related methods work in IE5+ and NN6. The page includes a simple, two-item list. A form enables you to add items to the end of the list or replace the last item with a different entry.

The `append()` function creates a new LI element and then uses the `appendChild()` method to attach the text box text as the displayed text for the item. The nested expression, `document.createTextNode(form.input.value)`, evaluates to a legitimate node that is appended to the new LI item. All of this occurs before the new LI item is added to the document. In the final statement of the function, `appendChild()` is invoked from the vantage point of the UL element — thus adding the LI element as a child node of the UL element.

Invoking the `replaceChild()` method in the `replace()` function utilizes some of the same code. The main difference is that the `replaceChild()` method requires a second parameter: a reference to the child element to be replaced. This demonstration replaces the final child node of the UL list, so the function takes advantage of the `lastChild` property of all elements to get a reference to that final nested child. That reference becomes the second parameter to `replaceChild()`.

Listing 15-21: **Various Child Methods**

```
<HTML>
<HEAD>
<TITLE>appendChild(), removeChild(), and replaceChild() Methods</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function append(form) {
    if (form.input.value) {
        var newItem = document.createElement("LI")
        newItem.appendChild(document.createTextNode(form.input.value))
        document.getElementById("myUL").appendChild(newItem)
    }
}

function replace(form) {
    if (form.input.value) {
        var newItem = document.createElement("LI")
        var lastChild = document.getElementById("myUL").lastChild
        newItem.appendChild(document.createTextNode(form.input.value))
        document.getElementById("myUL").replaceChild(newItem, lastChild)
    }
}

function restore() {
    var oneChild
    var mainObj = document.getElementById("myUL")
    while (mainObj.childNodes.length > 2) {
        oneChild = mainObj.lastChild
        mainObj.removeChild(oneChild)
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Child Methods</H1>
<HR>
Here is a list of items:
<UL ID="myUL"><LI>First Item
```

```
<LI>Second Item
</UL>
<FORM>
Enter some text to add/replace in the list:
<INPUT TYPE="text" NAME="input" SIZE=30><BR>
<INPUT TYPE="button" VALUE="Append to List" onClick="append(this.form)">
<INPUT TYPE="button" VALUE="Replace Final Item" onClick="replace(this.form)">
<INPUT TYPE="button" VALUE="Restore List" onClick="restore()">
</BODY>
</HTML>
```

The final part of the demonstration uses the `removeChild()` method to peel away all children of the UL element until just the two original items are left standing. Again, the `lastChild` property comes in handy as the `restore()` function keeps removing the last child until only two remain. Upon restoring the list, IE5/Mac fails to render the list bullets; but in the browser's object model, the UL element still exists.

applyElement(*elementObject*[, *type*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

To help you visualize the impact of the `applyElement()` method with its different parameter settings, Listing 15-22 enables you to apply a new element (an EM element) to a SPAN element inside a paragraph. At any time, you can view the HTML of the entire P element to see where the EM element is applied, as well as its impact on the element containment hierarchy for the paragraph.

After you load the page, inspect the HTML for the paragraph before doing anything else. Notice the SPAN element and its nested FONT element, both of which surround the one-word content. If you apply the EM element inside the SPAN element (click the middle button), the SPAN element's first (and only) child element becomes the EM element; the FONT element is now a child of the new EM element.

Listing 15-22: **Using the applyElement() Method**

```
<HTML>
<HEAD>
<TITLE>applyElement() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function applyOutside() {
    var newItem = document.createElement("EM")
    newItem.id = newItem.uniqueID
    document.all.mySpan.applyElement(newItem)
```

Continued

Listing 15-22 *(continued)*

```
}

function applyInside() {
    var newItem = document.createElement("EM")
    newItem.id = newItem.uniqueID
    document.all.mySpan.applyElement(newItem, "inside")
}

function showHTML() {
    alert(document.all.myP.outerHTML)
}
</SCRIPT>
</HEAD>
<BODY>
<H1>applyElement() Method</H1>
<HR>
<P ID="myP">A simple paragraph with a <SPAN ID="mySpan">
<FONT SIZE=+1>special</FONT></SPAN> word in it.</P>
<FORM>
<INPUT TYPE="button" VALUE="Apply <EM> Outside" onClick="applyOutside()">
<INPUT TYPE="button" VALUE="Apply <EM> Inside" onClick="applyInside()">
<INPUT TYPE="button" VALUE="Show <P> HTML..." onClick="showHTML()"><BR>
<INPUT TYPE="button" VALUE="Restore Paragraph" onClick="location.reload()">
</FORM>
</BODY>
</HTML>
```

The visible results of applying the EM element inside and outside the SPAN element in this case are the same. But you can see from the HTML results that each element impacts the element hierarchy quite differently.

attachEvent("*eventName*", *functionRef*)
detachEvent("*eventName*", *functionRef*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible)* to create an anonymous function that is called in response to an `onmousedown` event of the first paragraph on the page. Begin by assigning the anonymous function to global variable `a` (already initialized in The Evaluator) in the upper text box:

```
a = new Function("alert('Function created at " + (new Date()) + "')")
```

The quote marks and parentheses can get jumbled easily, so enter this expression carefully. When you enter the expression successfully, the Results box shows the function's text. Now assign this function to the `onmousedown` event of the `myP` element by entering the following statement into the upper text box:

```
document.all.myP.attachEvent("onmousedown", a)
```

The Results box displays `true` when successful. If you mouse down on the first paragraph, an alert box displays the date and time that the anonymous function was created (when the `new Date()` expression was evaluated).

Now, disconnect the event relationship from the object by entering the following statement into the upper text box:

```
document.all.myP.detachEvent("onmousedown", a)
```

blur()
focus()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

To show how both the `window.focus()` method and its opposite (`window.blur()`) operate, Listing 15-23 for NN3+ and IE4+ creates a two-window environment. From each window, you can bring the other window to the front. The main window uses the object returned by `window.open()` to assemble the reference to the new window. In the subwindow (whose content is created entirely on the fly by JavaScript), `self.opener` is summoned to refer to the original window, while `self.blur()` operates on the subwindow itself (except for the buggy behavior of NN6 noted earlier). Blurring one window and focusing on another window yields the same result of sending the window to the back of the pile.

Listing 15-23: The window.focus() and window.blur() Methods

```
<HTML>
<HEAD>
<TITLE>Window Focus() and Blur()</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
// declare global variable name
var newWindow = null
function makeNewWindow() {
    // check if window already exists
    if (!newWindow || newWindow.closed) {
        // store new window object in global variable
```

Continued

Listing 15-23 *(continued)*

```
        newWindow = window.open("","","width=250,height=250")
        // pause briefly to let IE3 window finish opening
        setTimeout("fillWindow()",100)
    } else {
        // window already exists, so bring it forward
        newWindow.focus()
    }
}
// assemble new content and write to subwindow
function fillWindow() {
    var newContent = "<HTML><HEAD><TITLE>Another Subwindow</TITLE></HEAD>"
    newContent += "<BODY bgColor='salmon'>"
    newContent += "<H1>A Salmon-Colored Subwindow.</H1>"
    newContent += "<FORM><INPUT TYPE='button' VALUE='Bring Main to Front'
onClick='self.opener.focus()'>"
    // the following button doesn't work in NN6
    newContent += "<FORM><INPUT TYPE='button' VALUE='Put Me in Back'
onClick='self.blur()'>"
    newContent += "</FORM></BODY></HTML>"
    // write HTML to new window document
    newWindow.document.write(newContent)
    newWindow.document.close()
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Window focus() and blur() Methods</H1>
<HR>
<FORM>
<INPUT TYPE="button" NAME="newOne" VALUE="Show New Window"
onClick="makeNewWindow()">
</FORM>
</BODY>
</HTML>
```

A key ingredient to the success of the `makeNewWindow()` function in Listing 15-23 is the first conditional expression. Because `newWind` is initialized as a `null` value when the page loads, that is its value the first time through the function. But after you open the subwindow the first time, `newWind` is assigned a value (the subwindow object) that remains intact even if the user closes the window. Thus, the value doesn't revert to `null` by itself. To catch the possibility that the user has closed the window, the conditional expression also sees if the window is closed. If it is, a new subwindow is generated, and that new window's reference value is reassigned to the `newWind` variable. On the other hand, if the window reference exists and the window is not closed, the `focus()` method brings that subwindow to the front. You can see the `focus()` method for a text object in action in *JavaScript Bible* Chapter 25's description of the `select()` method for text objects.

clearAttributes()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to examine the attributes of an element before and after you apply `clearAttributes()`. To begin, display the HTML for the table element on the page by entering the following statement into the upper text field:

```
myTable.outerHTML
```

Notice the attributes associated with the `<TABLE>` tag. Look at the rendered table to see how attributes such as `BORDER` and `WIDTH` affect the display of the table. Now, enter the following statement in the top text box to remove all removable attributes from this element:

```
myTable.clearAttributes()
```

First, look at the table. The border is gone, and the table is rendered only as wide as is necessary to display the content with no cell padding. Lastly, view the results of the `clearAttributes()` method in the `outerHTML` of the table again:

```
myTable.outerHTML
```

The source code file has not changed, but the object model in the browser's memory reflects the changes you made.

click()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `click()` method. The page includes various types of buttons at the bottom. You can "click" the checkbox, for example, by entering the following statement in the topmost text field:

```
document.myForm2.myCheckbox.click()
```

If you use a recent browser version, you most likely can see the checkbox change states between checked and unchecked each time you execute the statement.

cloneNode(*deepBoolean*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to clone, rename, and append an element found in The Evaluator's source code. Begin by cloning the paragraph element named `myP` along with all of its content. Enter the following statement into the topmost text field:

```
a = document.getElementById("myP").cloneNode(true)
```

The variable `a` now holds the clone of the original node, so you can change its `ID` attribute at this point by entering the following statement:

```
a.setAttribute("ID", "Dolly")
```

If you want to see the properties of the cloned node, enter `a` into the lower text field. The precise listing of properties you see depends on whether you use NN or IE; in either case, you should be able to locate the `id` property, whose value is now `Dolly`.

As a final step, append this newly named node to the end of the `body` element by entering the following statement into the topmost text field:

```
document.body.appendChild(a)
```

You can now scroll down to the bottom of the page and see a duplicate of the content. But because the two nodes have different `ID` attributes, they cannot confuse scripts that need to address one or the other.

componentFromPoint(*x*,*y*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

You can experiment with this method in the code supplied with Listing 15-24. As presented, the method is associated with a TEXTAREA object that is specifically sized to display both vertical and horizontal scrollbars. As you click various areas of the TEXTAREA and the rest of the page, the status bar displays information about the location of the event with the help of the `componentFromPoint()` method.

The script utilizes a combination of the `event.srcElement` property and the `componentFromPoint()` method to help you distinguish how you can use each one

for different types of event processing. The `srcElement` property is used initially as a filter to decide whether the status bar will reveal further processing about the TEXTAREA element's event details.

The `onMouseDown` event handler in the BODY element triggers all event processing. IE events bubble up the hierarchy (and no events are cancelled in this page), so all `mouseDown` events eventually reach the BODY element. Then, the `whereInWorld()` function can compare each `mouseDown` event from any element against the textarea's geography.

Listing 15-24: **Using the componentFromPoint() Method**

```
<HTML>
<HEAD>
<TITLE>componentFromPoint() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function whereInWorld(elem) {
    var x = event.clientX
    var y = event.clientY
    var component = document.all.myTextarea.componentFromPoint(x,y)
    if (window.event.srcElement == document.all.myTextarea) {
        if (component == "") {
            status = "mouseDown event occurred inside the element"
        } else {
            status = "mouseDown occurred on the element\'s " + component
        }
    } else {
        status = "mouseDown occurred " + component + " of the element"
    }
}
</SCRIPT>
</HEAD>
<BODY onMouseDown="whereInWorld()">
<H1>componentFromPoint() Method</H1>
<HR>
<P>Tracking the mouseDown event relative to the textarea object. View results in
status bar.</P>
<FORM>
<TEXTAREA NAME="myTextarea" WRAP="off" COLS=12 ROWS=4>
This is Line 1
This is Line 2
This is Line 3
This is Line 4
This is Line 5
This is Line 6
</TEXTAREA>
</FORM>
</BODY>
</HTML>
```

contains(*elementObjectReference*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Using The Evaluator (Chapter 13 in the *JavaScript Bible*), see how the `contains()` method responds to the object combinations in each of the following statements as you enter them into the upper text box:

```
document.body.contains(document.all.myP)
document.all.myP.contains(document.all.item("myEM"))
document.all.myEM.contains(document.all.myEM)
document.all.myEM.contains(document.all.myP)
```

Feel free to test other object combinations within this page.

detachEvent()

See `attachEvent()`.

dispatchEvent(*eventObject*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Listing 15-25 demonstrates the `dispatchEvent()` method as defined in the W3C DOM Level 2. The behavior is identical to that of Listing 15-26, which demonstrates the IE5.5 equivalent: `fireEvent()`. This example does not perform all intended actions in the first release of NN6 because the browser does not fully implement the `document.createEvent()` method. The example is designed to operate more completely in a future version that supports event generation.

Listing 15-25: **Using the dispatchEvent() Method**

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
#mySPAN {font-style:italic}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
// assemble a couple event object properties
```

```
function getEventProps(evt) {
    var msg = ""
    var elem = evt.target
    msg += "event.target.nodeName: " + elem.nodeName + "\n"
    msg += "event.target.parentNode: " + elem.parentNode.id + "\n"
    msg += "event button: " + evt.button
    return msg
}

// onClick event handlers for body, myP, and mySPAN
function bodyClick(evt) {
    var msg = "Click event processed in BODY\n\n"
    msg += getEventProps(evt)
    alert(msg)
    checkCancelBubble(evt)
}
function pClick(evt) {
    var msg = "Click event processed in P\n\n"
    msg += getEventProps(evt)
    alert(msg)
    checkCancelBubble(evt)
}
function spanClick(evt) {
    var msg = "Click event processed in SPAN\n\n"
    msg += getEventProps(evt)
    alert(msg)
    checkCancelBubble(evt)
}

// cancel event bubbling if check box is checked
function checkCancelBubble(evt) {
    if (document.controls.bubbleOn.checked) {
        evt.stopPropagation()
    }
}

// assign onClick event handlers to three elements
function init() {
    document.body.onclick = bodyClick
    document.getElementById("myP").onclick = pClick
    document.getElementById("mySPAN").onclick = spanClick
}

// invoke fireEvent() on object whose ID is passed as parameter
function doDispatch(objID, evt) {
    // don't let button clicks bubble
    evt.stopPropagation()
    var newEvt = document.createEvent("MouseEvent")
    if (newEvt) {
        newEvt.button = 3
        document.getElementById(objID).dispatchEvent(newEvt)
    } else {
```

Continued

Listing 15-25 *(continued)*

```
        alert("This browser version does not support the feature.")
    }
}
</SCRIPT>
</HEAD>
<BODY ID="myBODY" onLoad="init()">
<H1>fireEvent() Method</H1>
<HR>
<P ID="myP">This is a paragraph <SPAN ID="mySPAN">(with a nested SPAN)</SPAN>
that receives click events.</SPAN></P>
<HR>
<P><B>Control Panel</B></P>
<FORM NAME="controls">
<P><INPUT TYPE="checkbox" NAME="bubbleOn"
onClick="event.stopPropagation()">Cancel event bubbling.</P>
<P><INPUT TYPE="button" VALUE="Fire Click Event on BODY"
onClick="doDispatch('myBODY', event)"></P>
<P><INPUT TYPE="button" VALUE="Fire Click Event on myP"
onClick="doDispatch('myP', event)"></P>
<P><INPUT TYPE="button" VALUE="Fire Click Event on mySPAN"
onClick="doDispatch('mySPAN', event)"></P>
</FORM>
</BODY>
</HTML>
```

fireEvent("*eventType*"[, *eventObjectRef*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

The small laboratory of Listing 15-26 enables you to explore the possibilities of the IE5.5 `fireEvent()` method while reinforcing event bubbling concepts in IE. Three nested element objects are assigned separate `onClick` event handlers (via the `init()` function invoked after the page loads — although you can also set these event handlers via `onClick` attributes in the tags). Each handler displays an alert whose content reveals which object's event handler was triggered and the tag name and ID of the object that received the event. The default behavior of the page is to allow event bubbling, but a checkbox enables you to turn off bubbling.

After you load the page, click the italic segment (a nested SPAN element) to receive a series of three alert boxes. The first advises you that the SPAN element's `onClick` event handler is processing the event and that the SPAN element (whose ID

is mySPAN) is, indeed, the source element of the event. Because event bubbling is enabled by default, the event bubbles upward to the SPAN element's next outermost container: the myP paragraph element. (However, mySPAN is still the source element.) Finally, the event reaches the BODY element. If you click in the H1 element at the top of the page, the event is not processed until it reaches the BODY element — although the H1 element is the source element because that's what you clicked. In all cases, when you explicitly click something to generate the onclick event, the event's button property shows zero to signify the primary mouse button in IE.

Now onto the real purpose of this example: the fireEvent() method. Three buttons enable you to direct a click event to each of the three elements that have event handlers defined for them. The events fired this way are artificial, generated via the createEventObject() method. For demonstration purposes, the button property of these scripted events is set to 3. This property value is assigned to the event object that eventually gets directed to an element. With event bubbling left on, the events sent via fireEvent() behave just like the physical clicks on the elements. Similarly, if you disable event bubbling, the first event handler to process the event cancels bubbling, and no further processing of that event occurs. Notice that event bubbling is cancelled within the event handlers that process the event. To prevent the clicks of the checkbox and action buttons from triggering the BODY element's onClick event handlers, event bubbling is turned off for the buttons right away.

Listing 15-26: **Using the fireEvent() Method**

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
#mySPAN {font-style:italic}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
// assemble a couple event object properties
function getEventProps() {
    var msg = ""
    var elem = event.srcElement
    msg += "event.srcElement.tagName: " + elem.tagName + "\n"
    msg += "event.srcElement.id: " + elem.id + "\n"
    msg += "event button: " + event.button
    return msg
}

// onClick event handlers for body, myP, and mySPAN
function bodyClick() {
    var msg = "Click event processed in BODY\n\n"
    msg += getEventProps()
    alert(msg)
    checkCancelBubble()
}
function pClick() {
    var msg = "Click event processed in P\n\n"
    msg += getEventProps()
    alert(msg)
```

Continued

Listing 15-26 *(continued)*

```
     checkCancelBubble()
}
function spanClick() {
    var msg = "Click event processed in SPAN\n\n"
    msg += getEventProps()
    alert(msg)
    checkCancelBubble()
}

// cancel event bubbling if check box is checked
function checkCancelBubble() {
    event.cancelBubble = document.controls.bubbleOn.checked
}

// assign onClick event handlers to three elements
function init() {
    document.body.onclick = bodyClick
    document.all.myP.onclick = pClick
    document.all.mySPAN.onclick = spanClick
}

// invoke fireEvent() on object whose ID is passed as parameter
function doFire(objID) {
    var newEvt = document.createEventObject()
    newEvt.button = 3
    document.all(objID).fireEvent("onclick", newEvt)
    // don't let button clicks bubble
    event.cancelBubble = true
}
</SCRIPT>
</HEAD>
<BODY ID="myBODY" onLoad="init()">
<H1>fireEvent() Method</H1>
<HR>
<P ID="myP">This is a paragraph <SPAN ID="mySPAN">(with a nested SPAN)</SPAN>
that receives click events.</SPAN></P>
<HR>
<P><B>Control Panel</B></P>
<FORM NAME="controls">
<P><INPUT TYPE="checkbox" NAME="bubbleOn"
onClick="event.cancelBubble=true">Cancel event bubbling.</P>
<P><INPUT TYPE="button" VALUE="Fire Click Event on BODY"
onClick="doFire('myBODY')"></P>
<P><INPUT TYPE="button" VALUE="Fire Click Event on myP"
onClick="doFire('myP')"></P>
<P><INPUT TYPE="button" VALUE="Fire Click Event on mySPAN"
onClick="doFire('mySPAN')"></P>
</FORM>
</BODY>
</HTML>
```

focus()

See blur().

getAdjacentText("*position*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to examine all four adjacent text possibilities for the myP and nested myEM elements in that document. Enter each of the following statements into the upper text box, and view the results:

```
document.all.myP.getAdjacentText("beforeBegin")
document.all.myP.getAdjacentText("afterBegin")
document.all.myP.getAdjacentText("beforeEnd")
document.all.myP.getAdjacentText("afterEnd")
```

The first and last statements return empty strings because the myP element has no text fragments surrounding it. The afterBegin version returns the text fragment of the myP element up to, but not including, the EM element nested inside. The beforeEnd string picks up after the end of the nested EM element and returns all text to the end of myP.

Now, see what happens with the nested myEM element:

```
document.all.myEM.getAdjacentText("beforeBegin")
document.all.myEM.getAdjacentText("afterBegin")
document.all.myEM.getAdjacentText("beforeEnd")
document.all.myEM.getAdjacentText("afterEnd")
```

Because this element has no nested elements, the afterBegin and beforeEnd strings are identical: the same value as the innerText property of the element.

getAttribute("*attributeName*" [, *caseSensitivity*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the getAttribute() method for the elements in the page. For IE4, use the document.all

notation. IE5 and NN6 understand the W3C standard `getElementById()` method of addressing an element. You can enter the following sample statements into the top text box to view attribute values.

IE4:

```
document.all.myTable.getAttribute("width")
document.all.myTable.getAttribute("border")
```

IE5/NN6:

```
document.getElementById("myTable").getAttribute("width")
document.getElementById("myTable").getAttribute("border")
```

getAttributeNode("*attributeName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to explore the `getAttributeNode()` method in NN6. The Results TEXTAREA element provides several attributes to check out. Because the method returns an object, enter the following statements into the bottom text field so you can view the properties of the attribute node object returned by the method:

```
document.getElementById("output").getAttributeNode("COLS")
document.getElementById("output").getAttributeNode("ROWS")
document.getElementById("output").getAttributeNode("wrap")
document.getElementById("output").getAttributeNode("style")
```

All (except the last) statements display a list of properties for each attribute node object. The last statement, however, returns nothing because the `STYLE` attribute is not specified for the element.

getBoundingClientRect()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-27 employs both the `getBoundingClientRect()` and `getClientRects()` methods in a demonstration of how they differ. A set of elements is grouped within a SPAN element named `main`. The group consists of two paragraphs and an unordered list.

Two controls enable you to set the position of an underlying highlight rectangle to any line of your choice. A checkbox enables you to set whether the highlight rectangle should be only as wide as the line or the full width of the bounding rectangle for the entire SPAN element.

All the code is located in the `hilite()` function. The SELECT and checkbox elements invoke this function. Early in the function, the `getClientRects()` method is invoked for the `main` element to capture a snapshot of all `TextRectangle`s for the entire element. This array comes in handy when the script needs to get the coordinates of a rectangle for a single line, as chosen in the SELECT element.

Whenever the user chooses a number from the SELECT list and the value is less than the total number of `TextRectangle` objects in `clientRects`, the function begins calculating the size and location of the underlying yellow highlighter. When the Full Width checkbox is checked, the left and right coordinates are obtained from the `getBoundingClientRect()` method because the entire SPAN element's rectangle is the space you're interested in; otherwise, you pull the `left` and `right` properties from the chosen rectangle in the `clientRects` array.

Next comes the assignment of location and dimension values to the `hiliter` object's `style` property. The top and bottom are always pegged to whatever line is selected, so the `clientRects` array is polled for the chosen entry's `top` and `bottom` properties. The previously calculated `left` value is assigned to the `hiliter` object's `pixelLeft` property, while the width is calculated by subtracting the `left` from the `right` coordinates. Notice that the `top` and `left` coordinates also take into account any vertical or horizontal scrolling of the entire body of the document. If you resize the window to a smaller size, line wrapping throws off the original line count. However, an invocation of `hilite()` from the `onResize` event handler applies the currently chosen line number to whatever content falls in that line after resizing.

Listing 15-27: **Using getBoundingClientRect()**

```
<HTML>
<HEAD>
<TITLE>getClientRects() and getBoundClientRect() Methods</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function hilite() {
    var hTop, hLeft, hRight, hBottom, hWidth
    var select = document.forms[0].choice
    var n = parseInt(select.options[select.selectedIndex].value) - 1
    var clientRects = document.all.main.getClientRects()
    var mainElem = document.all.main
    if (n >= 0 && n < clientRects.length) {
        if (document.forms[0].fullWidth.checked) {
            hLeft = mainElem.getBoundingClientRect().left
            hRight = mainElem.getBoundingClientRect().right
        } else {
            hLeft = clientRects[n].left
            hRight = clientRects[n].right
        }
        document.all.hiliter.style.pixelTop = clientRects[n].top +
```

Continued

Listing 15-27 *(continued)*

```
            document.body.scrollTop
        document.all.hiliter.style.pixelBottom = clientRects[n].bottom
        document.all.hiliter.style.pixelLeft = hLeft + document.body.scrollLeft
        document.all.hiliter.style.pixelWidth = hRight - hLeft
        document.all.hiliter.style.visibility = "visible"
    } else if (n > 0) {
        alert("The content does not have that many lines.")
        document.all.hiliter.style.visibility = "hidden"
    }
}
</SCRIPT>
</HEAD>
<BODY onResize="hilite()">
<H1>getClientRects() and getBoundClientRect() Methods</H1>
<HR>
<FORM>
Choose a line to highlight:
<SELECT NAME="choice" onChange="hilite()">
<OPTION VALUE=0>
<OPTION VALUE=1>1
<OPTION VALUE=2>2
<OPTION VALUE=3>3
<OPTION VALUE=4>4
<OPTION VALUE=5>5
<OPTION VALUE=6>6
<OPTION VALUE=7>7
<OPTION VALUE=8>8
<OPTION VALUE=9>9
<OPTION VALUE=10>10
<OPTION VALUE=11>11
<OPTION VALUE=12>12
<OPTION VALUE=13>13
<OPTION VALUE=14>14
<OPTION VALUE=15>15
</SELECT><BR>

<INPUT NAME="fullWidth" TYPE="checkbox" onClick="hilite()">
Full Width (bounding rectangle)
</FORM>
<SPAN ID="main">
<P>Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim adminim veniam, quis nostrud exercitation ullamco:</P>
<UL>
<LI>laboris
<LI>nisi
<LI>aliquip ex ea commodo
</UL>
<P>Duis aute irure dolor in reprehenderit involuptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deseruntmollit
```

```
anim id est laborum Et harumd und lookum like Greek to me, dereud
facilis est er expedit distinct.</P>
</SPAN>
<DIV ID="hiliter"
STYLE="position:absolute; background-color:yellow; z-index:-1;
visibility:hidden">
</DIV>
</BODY>
</HTML>
```

Because the `z-index` style property of the `hiliter` element is set to `-1`, the element always appears beneath the primary content on the page. If the user selects a line number beyond the current number of lines in the main element, the `hiliter` element is hidden.

`getClientRects()`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listing 15-27, which demonstrates the differences between `getClientRects()` and `getBoundingClientRect()` and shows how you can use the two together.

`getElementsByTagName("tagName")`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `getElementsByTagName()` method. Enter the following statements one at a time into the upper text box and study the results:

```
document.body.getElementsByTagName("DIV")
document.body.getElementsByTagName("DIV").length
document.getElementById("myTable").getElementsByTagName("TD").length
```

Because the `getElementsByTagName()` method returns an array of objects, you can use one of those returned values as a valid element reference:

```
document.getElementsByTagName("FORM")[0].getElementsByTagName("INPUT").length
```

getExpression("*attributeName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listing 15-32 for the `setExpression()` method. This listing demonstrates the kinds of values returned by `getExpression()`.

hasChildNodes()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `hasChildNodes()` method. If you enter the following statement into the topmost text box:

```
document.getElementById("myP").hasChildNodes()
```

the returned value is `true`. You can find out how many nodes there are by getting the `length` of the `childNodes` array:

```
document.getElementById("myP").childNodes.length
```

This expression reveals a total of three nodes: the two text nodes and the EM element between them. Check out whether the first text node has any children:

```
document.getElementById("myP").childNodes[0].hasChildNodes()
```

The response is `false` because text fragments do not have any nested nodes. But check out the EM element, which is the second child node of the `myP` element:

```
document.getElementById("myP").childNodes[1].hasChildNodes()
```

The answer is `true` because the EM element has a text fragment node nested within it. Sure enough, the statement

```
document.getElementById("myP").childNodes[1].childNodes.length
```

yields a node count of `1`. You can also go directly to the EM element in your references:

```
document.getElementById("myEM").hasChildNodes()
document.getElementById("myEM").childNodes.length
```

If you want to see the properties of the text fragment node inside the EM element, enter the following into the lower text box:

```
document.getElementById("myEM").childNodes[0]
```

You can see that the `data` and `nodeValue` properties for the text fragment return the text `"all"`.

`insertAdjacentElement("location", elementObject)`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in *JavaScript Bible*) to experiment with the `insertAdjacentElement()` method. The goal of the experiment is to insert a new H1 element above the `myP` element.

All actions require you to enter a sequence of statements in the topmost text box. Begin by storing a new element in the global variable `a`:

```
a = document.createElement("H1")
```

Give the new object some text:

```
a.innerText = "New Header"
```

Now, insert this element before the start of the `myP` object:

```
myP.insertAdjacentElement("beforeBegin", a)
```

Notice that you have not assigned an `id` property value to the new element. But because the element was inserted by reference, you can modify the inserted object by changing the object stored in the `a` variable:

```
a.style.color = "red"
```

The inserted element is also part of the document hierarchy, so you can access it through hierarchy references such as `myP.previousSibling`.

The parent element of the newly inserted element is the BODY. Thus, you can inspect the current state of the HTML for the rendered page by entering the following statement into the topmost text box:

```
document.body.innerHTML
```

If you scroll down past the first form, you can find the `<H1>` element that you added along with the `STYLE` attribute.

```
insertAdjacentHTML("location", "HTMLtext")
insertAdjacentText("location", "text")
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with these two methods. The example here demonstrates the result of employing both methods in an attempt to add some HTML to the beginning of the `myP` element.

Begin by assigning a string of HTML code to the global variable `a`:

```
a = "<B ID='myB'>Important News!</B>"
```

Because this HTML is to go on the same line as the start of the `myP` paragraph, use the `afterBegin` parameter for the insert method:

```
myP.insertAdjacentHTML("afterBegin", a)
```

Notice that there is no space after the exclamation mark of the inserted HTML. But to prove that the inserted HTML is genuinely part of the document's object model, you can now insert the text of a space after the B element whose ID is `myB`:

```
myB.insertAdjacentText("afterEnd", " ")
```

Each time you evaluate the preceding statement (by repeatedly clicking the Evaluate button or pressing Enter with the cursor in the topmost field), an additional space is added.

You should also see what happens when the string to be inserted with `insertAdjacentText()` contains HTML tags. Reload The Evaluator and enter the following two statements into the topmost field, evaluating each one in turn:

```
a = "<B ID='myB'>Important News!</B>"
myP.insertAdjacentText("afterBegin", a)
```

The HTML is not interpreted but is displayed as plain text. There is no object named `myB` after executing this latest insert method.

```
insertBefore(newChildNodeObject[, referenceChildNode])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Listing 15-28 demonstrates how the `insertBefore()` method can insert child elements (LI) inside a parent (OL) at different locations, depending on the second parameter. A text box enables you to enter your choice of text and/or HTML for insertion at various locations within the OL element. If you don't specify a position, the second parameter of `insertBefore()` is passed as `null` — meaning that the new child node is added to the end of the existing children. But choose a spot from the select list where you want to insert the new item. The value of each SELECT list option is an index of one of the first three child nodes of the OL element.

Listing 15-28: **Using the insertBefore() Method**

```
<HTML>
<HEAD>
<TITLE>insertBefore() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doInsert(form) {
    if (form.newText) {
        var newChild = document.createElement("LI")
        newChild.innerHTML = form.newText.value
        var choice = form.itemIndex.options[form.itemIndex.selectedIndex].value
        var insertPoint = (isNaN(choice)) ?
            null : document.getElementById("myUL").childNodes[choice]
        document.getElementById("myUL").insertBefore(newChild, insertPoint)
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>insertBefore() Method</H1>
<HR>
<FORM onSubmit="return false">
<P>Enter text or HTML for a new list item:
<INPUT TYPE="text" NAME="newText" SIZE=40 VALUE=""></P>
<P>Before which existing item?
<SELECT NAME="itemIndex">
    <OPTION VALUE=null>None specified
    <OPTION VALUE=0>1
    <OPTION VALUE=1>2
    <OPTION VALUE=2>3
</SELECT></P>
<INPUT TYPE="button" VALUE="Insert Item" onClick="doInsert(this.form)">
</FORM>

<OL ID="myUL">
    <LI>Originally the First Item
    <LI>Originally the Second Item
    <LI>Originally the Third Item
</OL>
</BODY>
</HTML>
```

item(*index* | "*index*" [, *subIndex*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `item()` method. Type the following statements into the topmost text box and view the results for each:

NN6 and IE5

```
document.getElementById("myP").childNodes.length
document.getElementById("myP").childNodes.item(0).data
document.getElementById("myP").childNodes.item(1).nodeName
```

NN6, IE4, and IE5

```
document.forms[1].elements.item(0).type
```

IE4 and IE5

```
document.all.item("myP").outerHTML
myP.outerHTML
```

In the last two examples, both statements return the same string. The first example is helpful when your script is working with a string version of an object's name. If your script already knows the object reference, then the second approach is more efficient and compact.

mergeAttributes("*sourceObject*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-29 demonstrates the usage of `mergeAttributes()` in the process of replicating the same form input field while assigning a unique ID to each new field. So you can see the results as you go, I display the HTML for each input field in the field.

The `doMerge()` function begins by generating two new elements: a P and an INPUT element. Because these newly created elements have no properties associated with them, a unique ID is assigned to the INPUT element via the `uniqueID` property. Attributes from the field in the source code (`field1`) are merged into the new INPUT element. Thus, all attributes except `name` and `id` are copied to the new element. The INPUT element is inserted into the P element, and the P element is

appended to the document's form element. Finally, the `outerHTML` of the new element is displayed in its field. Notice that except for the `NAME` and `ID` attributes, all others are copied. This includes style sheet attributes and event handlers. To prove that the event handler works in the new elements, you can add a space to any one of them and press Tab to trigger the `onChange` event handler that changes the content to all uppercase characters.

Listing 15-29: **Using the mergeAttributes() Method**

```
<HTML>
<HEAD>
<TITLE>mergeAttributes() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doMerge(form) {
    var newPElem = document.createElement("P")
    var newInputElem = document.createElement("INPUT")
    newInputElem.id = newInputElem.uniqueID
    newInputElem.mergeAttributes(form.field1)
    newPElem.appendChild(newInputElem)
    form.appendChild(newPElem)
    newInputElem.value = newInputElem.outerHTML
}
// called by onChange event handler of fields
function upperMe(field) {
    field.value = field.value.toUpperCase()
}
</SCRIPT>
</HEAD>
<BODY onLoad="document.expandable.field1.value =
document.expandable.field1.outerHTML">
<H1>mergeAttributes() Method</H1>
<HR>
<FORM NAME="expandable" onSubmit="return false">
<P><INPUT TYPE="button" VALUE="Append Field 'Clone'"
onClick="doMerge(this.form)"></P>
<P><INPUT TYPE="text" NAME="field1" ID="FIELD1" SIZE=120 VALUE="" STYLE="font-
size:9pt" onChange="upperMe(this)"></P>
</FORM>
</BODY>
</HTML>
```

normalize()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator to experiment with the `normalize()` method in NN6. The following sequence adds a text node adjacent to one in the `myP` element. A subsequent invocation of the `normalize()` method removes the division between the adjacent text nodes.

Begin by confirming the number of child nodes of the `myP` element:

```
document.getElementById("myP").childNodes.length
```

Three nodes initially inhabit the element. Next, create a text node and append it as the last child of the `myP` element:

```
a = document.createTextNode("This means you!")
document.getElementById("myP").appendChild(a)
```

With the new text now rendered on the page, the number of child nodes increases to four:

```
document.getElementById("myP").childNodes.length
```

You can see that the last child node of `myP` is the text node you just created:

```
document.getElementById("myP").lastChild.nodeValue
```

But by invoking `normalize()` on `myP`, all adjacent text nodes are accumulated into single nodes:

```
document.getElementById("myP").normalize()
```

You can now see that the `myP` element is back to three child nodes, and the last child is a combination of the two previously distinct, but adjacent, text nodes:

```
document.getElementById("myP").childNodes.length
document.getElementById("myP").lastChild.nodeValue
```

releaseCapture()
setCapture(*containerBoolean*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-30 demonstrates the usage of `setCapture()` and `releaseCapture()` in a "quick-and-dirty" context menu for IE5+/Windows. The job of the context menu is to present a list of numbering styles for the ordered list of items on the page. Whenever the user brings up the context menu atop the OL element, the custom context menu appears. Event capture is turned on in the process to prevent mouse actions elsewhere on the page from interrupting the context menu choice. Even a click on the link set up as the title of the list is inhibited while the context menu is visible. A click anywhere outside of the context menu hides the menu. Clicking a

choice in the menu changes the `listStyleType` property of the OL object and hides the menu. Whenever the context menu is hidden, event capture is turned off so that clicking on the page (such as the link) works as normal.

For this design, `onClick`, `onMouseOver`, and `onMouseOut` event handlers are assigned to the DIV element that contains the context menu. To trigger the display of the context menu, the OL element has an `onContextMenu` event handler. This handler invokes the `showContextMenu()` function. In this function, event capture is assigned to the context menu DIV object. The DIV is also positioned at the location of the click before it is set to be visible. To prevent the system's regular context menu from also appearing, the `event` object's `returnValue` property is set to `false`. The context menu is shown activated in Figure 1-2.

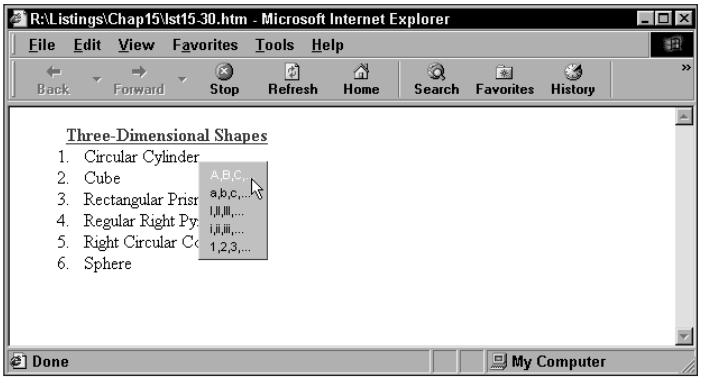


Figure 1-2: Displaying a customized context menu

Now that all mouse events on the page go through the `contextMenu` DIV object, let's examine what happens with different kinds of events triggered by user action. As the user rolls the mouse, a flood of `mouseover` and `mouseout` events fire. The event handlers assigned to the DIV manage these events. But notice that the two event handlers, `highlight()` and `unhighlight()`, perform action only when the `srcElement` property of the event is one of the menu items in the DIV. Because the page has no other `onMouseOver` or `onMouseOut` event handlers defined for elements up the containment hierarchy, you do not have to cancel event bubbling for these events.

When a user clicks the mouse button, different things happen depending on whether event capture is enabled. Without event capture, the `click` event bubbles up from wherever it occurred to the `onClick` event handler in the BODY element. (An alert dialog box displays to let you know when the event reaches the BODY.) But with event capture turned on (the context menu is showing), the `handleClick()` event handler takes over to apply the desired choice whenever the click is atop one of the context menu items. For all `click` events handled by this function, the context menu is hidden and the `click` event is canceled from bubbling up any higher (no alert dialog box appears). This takes place whether the user makes a choice in the context menu or clicks anywhere else on the page. In the latter case, all you need is for the context menu to go away like the real context menu does. For added insurance, the `onLoseCapture` event handler hides the context menu when a user performs any of the actions just listed that cancel capture.

Listing 15-30: **Using setCapture() and releaseCapture()**

```
<HTML>
<STYLE TYPE="text/css">
#contextMenu {position:absolute; background-color:#cfcfcf;
              border-style:solid; border-width:1px;
              border-color:#EFEFEF #505050 #505050 #EFEFEF;
              padding:3px 10px; font-size:8pt; font-family:Arial, Helvetica;
              line-height:150%; visibility:hidden}
.menuItem {color:black}
.menuItemOn {color:white}
OL {list-style-position:inside; font-weight:bold; cursor:nw-resize}
LI {font-weight:normal}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function showContextMenu() {
    contextMenu.setCapture()
    contextMenu.style.pixelTop = event.clientY + document.body.scrollTop
    contextMenu.style.pixelLeft = event.clientX + document.body.scrollLeft
    contextMenu.style.visibility = "visible"
    event.returnValue = false
}

function revert() {
    document.releaseCapture()
    hideMenu()
}

function hideMenu() {
    contextMenu.style.visibility = "hidden"
}

function handleClick() {
    var elem = window.event.srcElement
    if (elem.id.indexOf("menuItem") == 0) {
        shapesList.style.listStyleType = elem.LISTTYPE
    }
    revert()
    event.cancelBubble = true
}

function highlight() {
    var elem = event.srcElement
    if (elem.className == "menuItem") {
        elem.className = "menuItemOn"
    }
}

function unhighlight() {
    var elem = event.srcElement
    if (elem.className == "menuItemOn") {
```

```
        elem.className = "menuItem"
    }
}
</SCRIPT>
<BODY onClick="alert('You reached the document object.')" >
<OL ID="shapesList" onContextMenu="showContextMenu()">
<A HREF="javascript:alert('A sample link.')">Three-Dimensional Shapes</A>
<LI>Circular Cylinder</LI>
<LI>Cube</LI>
<LI>Rectangular Prism</LI>
<LI>Regular Right Pyramid</LI>
<LI>Right Circular Cone</LI>
<LI>Sphere</LI>
</OL>

<DIV ID="contextMenu" onLoseCapture="hideMenu()" onClick="handleClick()"
onMouseOver="highlight()" onMouseOut="unhighlight()">
<SPAN ID="menuItem1" CLASS="menuItem" LISTTYPE="upper-
alpha">A,B,C,...</SPAN><BR>
<SPAN ID="menuItem2" CLASS="menuItem" LISTTYPE="lower-
alpha">a,b,c,...</SPAN><BR>
<SPAN ID="menuItem3" CLASS="menuItem" LISTTYPE="upper-
roman">I,II,III,...</SPAN><BR>
<SPAN ID="menuItem4" CLASS="menuItem" LISTTYPE="lower-
roman">i,ii,iii,...</SPAN><BR>
<SPAN ID="menuItem5" CLASS="menuItem" LISTTYPE="decimal">1,2,3,...</SPAN><BR>
</DIV>
</BODY>
</HTML>
```

removeAttribute("*attributeName*" [, *caseSensitivity*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `removeAttribute()` method for the elements in the page. See the examples for the `setAttribute()` method later in this chapter, and enter the corresponding `removeAttribute()` statements in the top text box. Interlace statements using `getAttribute()` to verify the presence or absence of each attribute.

removeAttributeNode(*attributeNode*)
setAttributeNode(*attributeNode*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `setAttributeNode()` and `removeAttributeNode()` methods for the P element in the page. The task is to create and add a `STYLE` attribute to the P element. Begin by creating a new attribute and storing it temporarily in the global variable `a`:

```
a = document.createAttribute("style")
```

Assign a value to the `attribute` object:

```
a.nodeValue = "color:red"
```

Now insert the new attribute into the P element:

```
document.getElementById("myP").setAttributeNode(a)
```

The paragraph changes color in response to the newly added attribute.

Due to the NN6 bug that won't allow the method to return a reference to the newly inserted attribute node, you can artificially obtain such a reference:

```
b = document.getElementById("myP").getAttributeNode("style")
```

Finally, use the reference to the newly added attribute to remove it from the element:

```
document.getElementById("myP").removeAttribute(b)
```

Upon removing the attribute, the paragraph resumes its initial color. See the example for the `setAttribute()` method later in this chapter to discover how you can perform this same kind of operation with `setAttribute()`.

removeBehavior(*ID*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listings 15-19a and 15-19b earlier in this chapter for examples of how to use `addBehavior()` and `removeBehavior()`.

removeChild(*nodeObject*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

You can see an example of `removeChild()` as part of Listing 15-21 earlier in this chapter.

removeEventListener()

See `addEventListener()`.

removeExpression("*propertyName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

You can experiment with all three expression methods in The Evaluator (Chapter 13 in the *JavaScript Bible*). The following sequence adds an expression to a style sheet property of the `myP` element on the page and then removes it.

To begin, enter the number `24` in the bottom one-line text box in The Evaluator (but don't press Enter or click the List Properties button). This is the value used in the expression to govern the `fontSize` property of the `myP` object. Next, assign an expression to the `myP` object's `style` object by entering the following statement into the topmost text box:

```
myP.style.setExpression("fontSize","document.forms[0].inspector.value","JScript")
```

You can now enter different font sizes into the lower text box and have the values immediately applied to the `fontSize` property. (Keyboard events in the text box automatically trigger the recalculation.) The default unit is `px`, but you can also append other units (such as `pt`) to the value in the text field to see how different measurement units influence the same numeric value.

Before proceeding to the next step, enter a value other than 16 (the default `fontSize` value). Finally, enter the following statement in the topmost text box to disconnect the expression from the property:

```
myP.style.removeExpression("fontSize")
```

Notice that although you can no longer adjust the font size from the lower text box, the most recent value assigned to it still sticks to the element. To prove it, enter the following statement in the topmost text box to see the current value:

```
myP.style.fontSize
```

removeNode(*removeChildrenFlag*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Examine Listing 15-21 for the `appendChild()` method to understand the difference between `removeChild()` and `removeNode()`. In the `restore()` function, you can replace this statement

```
mainObj.removeChild(oneChild)
```

in IE5+ with

```
oneChild.removeNode(true)
```

The difference is subtle, but it is important to understand. See Listing 15-31 later in this chapter for another example of the `removeNode()` method.

replaceAdjacentText("*location*", "*text*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `replaceAdjacentText()` method. Enter each of the following statements into the top text box and watch the results in the `myP` element (and its nested `myEM` element) below the solid rule:

```
document.all.myEM.replaceAdjacentText("afterBegin", "twenty")
```

Notice that the `myEM` element's new text picks up the behavior of the element. In the meantime, the replaced text (`all`) is returned by the method and displayed in the Results box.

```
document.all.myEM.replaceAdjacentText("beforeBegin", "We need ")
```

All characters of the text fragment, including spaces, are replaced. Therefore, you may need to supply a trailing space, as shown here, if the fragment you replace has a space.

```
document.all.myP.replaceAdjacentText("beforeEnd", " good people.")
```

This is another way to replace the text fragment following the `myEM` element, but it is also relative to the surrounding `myP` element. If you now attempt to replace text after the end of the `myP` block-level element,

```
document.all.myP.replaceAdjacentText("afterEnd", "Hooray!")
```

the text fragment is inserted after the end of the `myP` element's tag set. The fragment is just kind of floating in the document object model as an unlabeled text node.

replaceChild(*newNodeObject, oldNodeObject*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

You can see an example of `replaceChild()` as part of Listing 15-21 earlier in this chapter.

replaceNode("*newNodeObject*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-31 demonstrates three node-related methods: `removeNode()`, `replaceNode()`, and `swapNode()`. These methods work in IE5+ only.

The page rendered from Listing 15-31 begins with a UL type list of four items. Four buttons control various aspects of the node structure of this list element. The first button invokes the `replace()` function, which changes the UL type to OL. To do this, the function must temporarily tuck away all child nodes of the original UL element so that they can be added back into the new OL element. At the same time, the old UL node is stored in a global variable (`oldNode`) for restoration in another function.

To replace the UL node with an OL, the `replace()` function creates a new, empty OL element and assigns the `myOL` ID to it. Next, the children (LI elements) are stored en masse as an array in the variable innards. The child nodes are then inserted into the empty OL element, using the `insertBefore()` method. Notice that as each child element from the `innards` array is inserted into the OL element, the child element is removed from the `innards` array. That's why the loop to insert the child nodes is a `while` loop that constantly inserts the first item of the `innards` array to

the new element. Finally, the `replaceNode()` method puts the new node in the old node's place, while the old node (just the UL element) is stored in `oldNode`.

The `restore()` function operates in the inverse direction of the `replace()` function. The same juggling of nested child nodes is required.

The third button invokes the `swap()` function, whose script exchanges the first and last nodes. The `swapNode()` method, like the others in this discussion, operates from the point of view of the node. Therefore, the method is attached to one of the swapped nodes, while the other node is specified as a parameter. Because of the nature of the OL element, the number sequence remains fixed but the text of the LI node swaps.

To demonstrate the `removeNode()` method, the fourth function removes the last child node of the list. Each call to `removeNode()` passes the `true` parameter to guarantee that the text nodes nested inside each LI node are also removed. Experiment with this method by setting the parameter to `false` (the default). Notice how the parent–child relationship changes when you remove the LI node.

Listing 15-31: **Using Node-Related Methods**

```
<HTML>
<HEAD>
<TITLE>removeNode(), replaceNode(), and swapNode() Methods</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// store original node between changes
var oldNode

// replace UL node with OL
function replace() {
    if (document.all.myUL) {
        var newNode = document.createElement("OL")
        newNode.id = "myOL"
        var innards = document.all.myUL.children
        while (innards.length > 0) {
            newNode.insertBefore(innards[0])
        }
        oldNode = document.all.myUL.replaceNode(newNode)
    }
}

// restore OL to UL
function restore() {
    if (document.all.myOL && oldNode) {
        var innards = document.all.myOL.children
        while (innards.length > 0) {
            oldNode.insertBefore(innards[0])
        }
        document.all.myOL.replaceNode(oldNode)

    }
}

// swap first and last nodes
function swap() {
```

```
    if (document.all.myUL) {
        document.all.myUL.firstChild.swapNode(document.all.myUL.lastChild)
    }
    if (document.all.myOL) {
        document.all.myOL.firstChild.swapNode(document.all.myOL.lastChild)
    }
}

// remove last node
function remove() {
    if (document.all.myUL) {
        document.all.myUL.lastChild.removeNode(true)
    }
    if (document.all.myOL) {
        document.all.myOL.lastChild.removeNode(true)
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Node Methods</H1>
<HR>
Here is a list of items:
<UL ID="myUL">
<LI>First Item
<LI>Second Item
<LI>Third Item
<LI>Fourth Item
</UL>
<FORM>
<INPUT TYPE="button" VALUE="Change to OL List" onClick="replace()">  
<INPUT TYPE="button" VALUE="Restore LI List" onClick="restore()">  
<INPUT TYPE="button" VALUE="Swap First/Last" onClick="swap()">  
<INPUT TYPE="button" VALUE="Remove Last" onClick="remove()">
</BODY>
</HTML>
```

You can accomplish the same functionality shown in Listing 15-31 in a cross-browser fashion using the W3C DOM. In place of the `removeNode()` and `replaceNode()` methods, use `removeChild()` and `replaceChild()` methods to shift the point of view (and object references) to the parent of the UL and OL objects: the `document.body`. Also, you need to change the `document.all` references to `document.getElementById()`.

scrollIntoView(*topAlignFlag*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `scrollIntoView()` method. Resize the browser window height so that you can see only the topmost text box and the Results textarea. Enter each of the following statements into the top text box and see where the `myP` element comes into view:

```
myP.scrollIntoView()
myP.scrollIntoView(false)
```

Expand the height of the browser window until you can see part of the table lower on the page. If you enter

```
myTable.scrollIntoView(false)
```

into the top text box, the page scrolls to bring the bottom of the table to the bottom of the window. But if you use the default parameter (`true` or empty),

```
myTable.scrollIntoView()
```

the page scrolls as far as it can in an effort to align the top of the element as closely as possible to the top of the window. The page cannot scroll beyond its normal scrolling maximum (although if the element is a positioned element, you can use dynamic positioning to place it wherever you want — including "off the page"). Also, if you shrink the window and try to scroll the top of the table to the top of the window, be aware that the TABLE element contains a CAPTION element so the caption is flush with the top of the window.

setActive()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to compare the `setActive()` and `focus()` methods. With the page scrolled to the top and the window sized so that you cannot see the sample check box near the bottom of the page, enter the following statement into the top text box:

```
document.forms[1].myCheckbox.setActive()
```

Scroll down to see that the checkbox has operational focus (press the spacebar to see). Now, scroll back to the top and enter the following:

```
document.forms[1].myCheckbox.focus()
```

This time, the checkbox gets focus and the page automatically scrolls the object into view.

setAttribute("*attributeName*", *value* [, *caseSensitivity*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `setAttribute()` method for the elements in the page. For IE4, use the `document.all` notation; IE5 and NN6 understand the W3C standard `getElementById()` method of addressing an element.

Setting attributes can have immediate impact on the layout of the page (just as setting an object's properties can). Enter these sample statements into the top text box to view attribute values:

IE4+:

```
document.all.myTable.setAttribute("width", "80%")
document.all.myTable.setAttribute("border", "5")
```

IE5+/NN6:

```
document.getElementById("myTable").setAttribute("width", "80%")
document.getElementById("myTable").setAttribute("border", "5")
```

setAttributeNode()

See `removeAttributeNode()`.

setCapture(*containerBoolean*)

See `releaseCapture()`.

setExpression("*propertyName*", "*expression*","*language*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-32 shows the `setExpression()`, `recalc()`, and `getExpression()` methods at work in a DHTML-based clock. Figure 1-3 shows the clock. As time clicks

by, the bars for hours, minutes, and seconds adjust their widths to reflect the current time. At the same time, the `innerHTML` of SPAN elements to the right of each bar display the current numeric value for the bar.

The dynamically calculated values in this example are based on the creation of a new date object over and over again to get the current time from the client computer clock. It is from the date object (stored in the variable called `now`) that the hour, minute, and second values are retrieved. Some other calculations are involved so that a value for one of these time components is converted into a pixel value for the width of the bars. The bars are divided into 24 (for the hours) and 60 (for the minutes and seconds) parts, so the scale for the two types differs. For the 60-increment bars in this application, each increment is set to 5 pixels (stored in `shortWidth`); the 24-increment bars are 2.5 times the `shortWidth`.

As the document loads, the three SPAN elements for the colored bars are given no width, which means that they assume the default width of zero. But after the page loads, the `onLoad` event handler invokes the `init()` function, which sets the initial values for each bar's width and the text (`innerHTML`) of the three labeled spans. Once these initial values are set, the `init()` function invokes the `updateClock()` function.

In the `updateClock()` function, a new date object is created for the current instant. The `document.recalc()` method is called, instructing the browser to recalculate the expressions that were set in the `init()` function and assign the new values to the properties. To keep the clock "ticking," the `setTimeout()` method is set to invoke this same `updateClock()` function in one second.

To see what the `getExpression()` method does, you can click the button on the page. It simply displays the returned value for one of the attributes that you assign using `setExpression()`.

Listing 15-32: **Dynamic Properties**

```
<HTML>
<HEAD>
<TITLE>getExpression(), setExpression(), and recalc() Methods</TITLE>
<STYLE TYPE="text/css">
TH {text-align:right}
SPAN {vertical-align:bottom}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">

var now = new Date()
var shortWidth = 5
var multiple = 2.5

function init() {
    with (document.all) {
        hoursBlock.style.setExpression("width",
            "now.getHours() * shortWidth * multiple","jscript")
        hoursLabel.setExpression("innerHTML",
            "now.getHours()","jscript")
        minutesBlock.style.setExpression("width",
            "now.getMinutes() * shortWidth","jscript")
```

```
        minutesLabel.setExpression("innerHTML",
            "now.getMinutes()","jscript")
        secondsBlock.style.setExpression("width",
            "now.getSeconds() * shortWidth","jscript")
        secondsLabel.setExpression("innerHTML",
            "now.getSeconds()","jscript")
    }

    updateClock()
}

function updateClock() {
    now = new Date()
    document.recalc()
    setTimeout("updateClock()",1000)
}

function showExpr() {
    alert("Expression for the \'Hours\' innerHTML property is:\r\n" +
document.all.hoursLabel.getExpression("innerHTML") + ".")
}
</SCRIPT>
</HEAD>
<BODY onLoad="init()">
<H1>getExpression(), setExpression(), recalc() Methods</H1>
<HR>
<P>This clock uses Dynamic Properties to calculate bar width and time
numbers:</P>
<TABLE BORDER=0>
<TR>
    <TH>Hours:</TH>
    <TD><SPAN ID="hoursBlock" STYLE="background-color:red"></SPAN>
 <SPAN ID="hoursLabel"></SPAN></TD>
</TR>
<TR>
    <TH>Minutes:</TH>
    <TD><SPAN ID="minutesBlock" STYLE="background-color:yellow"></SPAN>
 <SPAN ID="minutesLabel"></SPAN></TD>
</TR>
<TR>
    <TH>Seconds:</TH>
    <TD><SPAN ID="secondsBlock" STYLE="background-color:green"></SPAN>
 <SPAN ID="secondsLabel"></SPAN></TD>
</TR>
</TABLE>
<HR>
<FORM>
<INPUT TYPE="button" VALUE="Show 'Hours' number innerHTML Expression"
onClick="showExpr()"
</FORM>
</BODY>
</HTML>
```

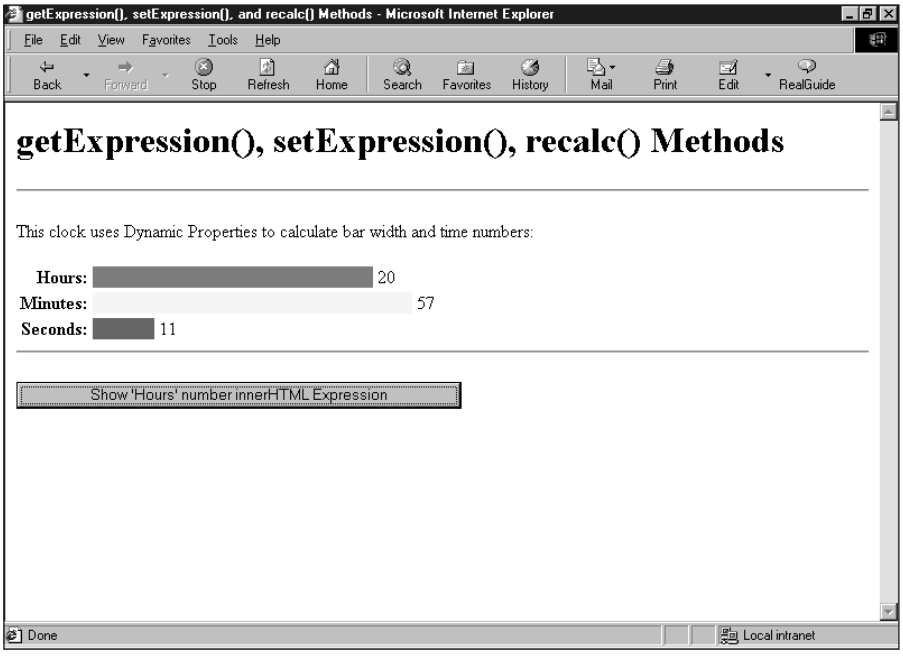


Figure 1-3: A clock controlled by dynamic properties

swapNode(*otherNodeObject*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listing 15-31 (the `replaceNode()` method) for an example of the `swapNode()` method in action.

tags("*tagName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `tags()` method. Enter the following statements one at a time into the upper text box and study the results:

```
document.all.tags("DIV")
document.all.tags("DIV").length
myTable.all.tags("TD").length
```

Because the `tags()` method returns an array of objects, you can use one of those returned values as a valid element reference:

```
document.all.tags("FORM")[1].elements.tags("INPUT").length
```

urns("*behaviorURN*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

In case the `urns()` method is reconnected in the future, you can add a button and function to Listing 15-19b that reveals whether the `makeHot.htc` behavior is attached to the `myP` element. Such a function looks like this:

```
function behaviorAttached() {
    if (document.all.urns("makeHot")) {
        alert("There is at least one element set to \'makeHot\'.")
    }
}
```

Event handlers

onActivate
onBeforeDeactivate
onDeactivate

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

You can modify Listing 15-34 later in this chapter by substituting `onActivate` for `onFocus` and `onDeactivate` for `onBlur`.

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `onBeforeDeactivate` event handler. To begin, set the `myP` element so it can accept focus:

```
myP.tabIndex = 1
```

If you repeatedly press the Tab key, the `myP` paragraph will eventually receive focus — indicated by the dotted rectangle around it. To see how you can prevent the element from losing focus, assign an anonymous function to the `onBeforeDeactivate` event handler, as shown in the following statement:

```
myP.onbeforedeactivate = new Function("event.returnValue=false")
```

Now you can press Tab all you like or click other focusable elements all you like, and the `myP` element will not lose focus until you reload the page (which clears away the event handler). Please do not do this on your pages unless you want to infuriate and alienate your site visitors.

onBeforeCopy

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

You can use the `onBeforeCopy` event handler to preprocess information prior to an actual copy action. In Listing 15-33, the function invoked by the second paragraph element's `onBeforeCopy` event handler selects the entire paragraph so that the user can select any character(s) in the paragraph to copy the entire paragraph into the clipboard. You can paste the results into the textarea to verify the operation. By assigning the paragraph selection to the `onBeforeCopy` event handler, the page notifies the user about what the copy operation will entail prior to making the menu choice. Had the operation been deferred to the `onCopy` event handler, the selection would have been made after the user chose Copy from the menu.

Listing 15-33: **The onBeforeCopy Event Handler**

```
<HTML>
<HEAD>
<TITLE>onBeforeCopy Event Handler</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function selectWhole() {
    var obj = window.event.srcElement
    var range = document.body.createTextRange()
    range.moveToElementText(obj)
    range.select()
    event.returnValue = false
}
</SCRIPT>
```

```
</HEAD>
<BODY>
<H1>onBeforeCopy Event Handler</H1>
<HR>
<P>Select one or more characters in the following paragraph. Then
execute a Copy command via Edit or context menu.</P>
<P ID="myP" onBeforeCopy="selectWhole()">Lorem ipsum dolor sit amet,
consectetaur adipisicing elit, sed do eiusmod tempor incididunt ut
labore et dolore magna aliqua. Ut enim adminim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</P>
<FORM>
<P>Paste results here:<BR>
<TEXTAREA NAME="output" COLS="60" ROWS="5"></TEXTAREA>
</P>
</FORM>
</BODY>
</HTML>
```

onBeforeCut

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

You can use the `onBeforeCut` event handler to preprocess information prior to an actual cut action. You can try this by editing a copy of Listing 15-33, changing the `onBeforeCopy` event handler to `onBeforeCut`. Notice that in its original form, the example does not activate the Cut item in either the context or Edit menu when you select some text in the second paragraph. But by assigning a function to the `onBeforeCut` event handler, the menu item is active, and the entire paragraph is selected from the function that is invoked.

onBeforeDeactivate

See `onActivate`.

onBeforeEditFocus

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator to explore the `onBeforeEditFocus` in IE5.5+. In the following sequence, you assign an anonymous function to the `onBeforeEditFocus` event handler of the `myP` element. The function turns the text color of the element to red when the event handler fires:

```
myP.onbeforeeditfocus = new Function("myP.style.color='red'")
```

Now turn on content editing for the `myP` element:

```
myP.contentEditable = true
```

If you now click inside the `myP` element on the page to edit its content, the text turns to red before you begin editing. In a page scripted for this kind of user interface, you would include some control that turns off editing and changes the color to normal.

If you wish to learn more about HTML content editing via the DHTML Editing ActiveX control, visit `http://msdn.microsoft.com/workshop/browser/mshtml/`.

onBeforePaste

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listing 15-45 for the `onPaste` event handler (later in this chapter) to see how the `onBeforePaste` and `onPaste` event handlers work together.

onBlur

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

More often than not, a page author uses the `onBlur` event handler to exert extreme control over the user, such as preventing a user from exiting out of a text box unless that user types something into the box. This is not a Web-friendly practice, and it is one that I discourage because there are intelligent ways to ensure a field has something typed into it before a form is submitted (see Chapter 43 of the *JavaScript Bible*). Listing 15-34 simply demonstrates the impact of the `TABINDEX` attribute in an IE5/Windows element with respect to the `onBlur` and `onFocus` events. Notice that as you press the Tab key, only the second paragraph issues the events even though all three paragraphs have event handlers assigned to them.

Listing 15-34: **onBlur and onFocus Event Handlers**

```
<HTML>
<HEAD>
<TITLE>onBlur and onBlur Event Handlers</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function showBlur() {
    var id = event.srcElement.id
    alert("Element \"" + id + "\" has blurred.")
}
function showFocus() {
    var id = event.srcElement.id
    alert("Element \"" + id + "\" has received focus.")
}
</SCRIPT>
</HEAD>
<BODY>
<H1 ID="H1" TABINDEX=2>onBlur and onBlur Event Handlers</H1>
<HR>
<P ID="P1" onBlur="showBlur()" onFocus="showFocus()">Lorem ipsum
dolor sit amet, consectetaur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim adminim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat.</P>

<P ID="P2" TABINDEX=1 onBlur="showBlur()" onFocus="showFocus()">Bis
nostrud exercitation ullam mmodo consequet. Duis aute involuptate
velit esse cillum dolore eu fugiat nulla pariatur. At vver eos et
accusam dignissum qui blandit est praesent luptatum delenit
aigueexcepteur sint occae.</P>

<P ID="P3" onBlur="showBlur()" onFocus="showFocus()">Unte af phen
neigepheings atoot Prexs eis phat eit sakem eit vory gast te Plok
peish ba useing phen roxas. Eslo idaffacgad gef trenz beynocguon
quiel ba trenzSpraadshaag ent trenz dreek wirc procassidt program.</P>

</BODY>
</HTML>
```

onClick

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The `onClick` event handler is one of the simplest to grasp and use. Listing 15-35 demonstrates its interaction with the `onDblClick` event handler and shows you how to prevent a link's intrinsic action from activating when combined with `click` events. As you click and/or double-click the link, the status bar displays a message associated with each event. Notice that if you double-click, the `click` event fires first with the first message immediately replaced by the second. For demonstration purposes, I show both backward-compatible ways of cancelling the link's intrinsic action. In practice, decide on one style and stick with it.

Listing 15-35: **Using onClick and onDblClick Event Handlers**

```
<HTML>
<HEAD>
<TITLE>onClick and onDblClick Event Handlers</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var msg = ""
function showClick() {
    msg = "The element has been clicked. "
    status = msg
}
function showDblClick() {
    msg = "The element has been double-clicked."
    status = msg
    return false
}
</SCRIPT>
</HEAD>
<BODY>
<H1>onClick and onDblClick Event Handlers</H1>
<HR>
<A HREF="#" onClick="showClick();return false"
onDblClick="return showDblClick()">
A sample link.</A>
</BODY>
</HTML>
```

onContextMenu

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listing 15-30 earlier in this chapter for an example of using the `onContextMenu` event handler with a custom context menu.

onCopy
onCut

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-36 shows both the `onBeforeCut` and `onCut` event handlers in action (as well as `onBeforePaste` and `onPaste`). Notice how the `handleCut()` function not only stuffs the selected word into the `clipboardData` object, but it also erases the selected text from the table cell element from where it came. If you replace the `onBeforeCut` and `onCut` event handlers with `onBeforeCopy` and `onCopy` (and change `handleCut()` to not eliminate the inner text of the event source element), the operation works with copy and paste instead of cut and paste. I demonstrate this later in the chapter in Listing 15-45.

Listing 15-36: Cutting and Pasting under Script Control

```
<HTML>
<HEAD>
<TITLE>onBeforeCut and onCut Event Handlers</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
TH {text-decoration:underline}
.blanks {text-decoration:underline}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function selectWhole() {
    var obj = window.event.srcElement
    var range = document.body.createTextRange()
    range.moveToElementText(obj)
    range.select()
    event.returnValue = false
}
function handleCut() {
    var rng = document.selection.createRange()
    clipboardData.setData("Text",rng.text)
    var elem = event.srcElement
```

Continued

Listing 15-36 *(continued)*

```
    elem.innerText = ""
    event.returnValue = false
}

function handlePaste() {
    var elem = window.event.srcElement
    if (elem.className == "blanks") {
        elem.innerHTML = clipboardData.getData("Text")
    }
    event.returnValue = false
}
function handleBeforePaste() {
    var elem = window.event.srcElement
    if (elem.className == "blanks") {
        event.returnValue = false
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>onBeforeCut and onCut Event Handlers</H1>
<HR>
<P>Your goal is to cut and paste one noun and one
adjective from the following table into the blanks
of the sentence. Select a word from the table and
use the Edit or context menu to cut it from the table.
Select one or more spaces of the blanks in the
sentence and choose Paste to replace the blank with
the clipboard contents.</P>

<TABLE CELLPADDING=5 onBeforeCut="selectWhole()" onCut="handleCut()" >
<TR><TH>Nouns</TH><TH>Adjectives</TH></TR>
<TR><TD>truck</TD><TD>round</TD></TR>
<TR><TD>doll</TD><TD>red</TD></TR>
<TR><TD>ball</TD><TD>pretty</TD></TR>
</TABLE>

<P ID="myP" onBeforePaste="handleBeforePaste()" onPaste="handlePaste()">
Pat said, "Oh my, the <SPAN ID="blank1" CLASS="blanks">
     </SPAN>
is so <SPAN ID="blank2" CLASS="blanks">
     </SPAN>!"</P>

<BUTTON onClick="location.reload()">Reset</BUTTON>
</BODY>
</HTML>
```

onDblClick

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

See Listing 15-35 (for the `onClick` event handler) to see the `onDblClick` event in action.

onDrag

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-37 shows several drag-related event handlers in action. The page resembles the example in Listing 15-36, but the scripting behind the page is quite different. In this example, the user is encouraged to select individual words from the Nouns and Adjectives columns and drag them to the blanks of the sentence. To beef up the demonstration, Listing 15-37 shows you how to pass the equivalent of array data from a drag source to a drag target. At the same time, the user has a fixed amount of time (two seconds) to complete each drag operation.

The `onDragStart` and `onDrag` event handlers are placed in the `<BODY>` tag because those events bubble up from any element that the user tries to drag. The scripts invoked by these event handlers filter the events so that the desired action is triggered only by the "hot" elements inside the table. This approach to event handlers prevents you from having to duplicate event handlers (or IE `<SCRIPT FOR=>` tags) for each table cell.

The `onDragStart` event handler invokes `setupDrag()`. This function cancels the `onDragStart` event except when the target element (in other words, the one about to be dragged) is one of the TD elements inside the table. To make this application smarter about what kind of word is dragged to which blank, it passes not only the word's text, but also some extra information about the word. This lets another event handler verify that a noun has been dragged to the first blank, while an adjective has been dragged to the second blank. To help with this effort, class names are assigned to the TD elements to distinguish the words from the Nouns column from the words of the Adjectives column. The `setupDrag()` function generates an array consisting of the `innerText` of the event's source element plus the element's class name. But the `event.dataTransfer` object cannot store array data types, so the `Array.join()` method converts the array to a string with a colon separating the entries. This string, then, is stuffed into the `event.dataTransfer` object. The object is instructed to render the cursor display during the drag-and-drop operation so that when the cursor is

atop a drop target, the cursor is the "copy" style. Figure 1-4 shows the cursor effect as the user drags a selected word from the columns to a blank field that is scripted as a drop target. Finally, the `setupDrag()` function is the first to execute in the drag operation, so a timer is set to the current clock time to time the drag operation.

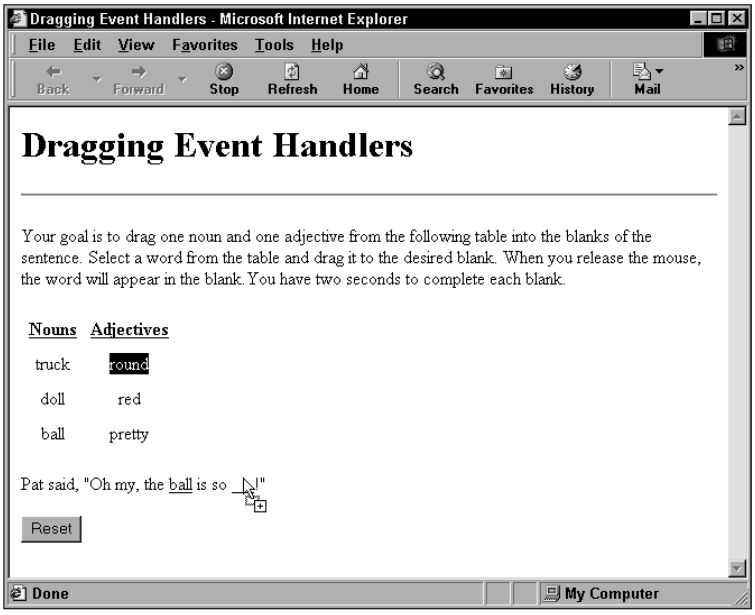


Figure 1-4: The cursor turns to a "copy" icon atop a designated drop target

The `onDrag` event handler (in the BODY) captures the `onDrag` events that are generated by whichever table cell element is the source element for the action. Each time the event fires (which is a lot during the action), the `timeIt()` function is invoked to compare the current time against the reference time (global `timer`) set when the drag starts. If the time exceeds two seconds (2,000 milliseconds), an alert dialog box notifies the user. To close the alert dialog box, the user must unclick the mouse button to end the drag operation.

To turn the blank SPAN elements into drop targets, their `onDragEnter`, `onDragOver`, and `onDrop` event handlers must set `event.returnValue` to `false`; also, the `event.dataTransfer.dropEffect` property should be set to the desired effect (`copy` in this case). These event handlers are placed in the P element that contains the two SPAN elements, again for simplicity. Notice, however, that the `cancelDefault()` functions do their work only if the target element is one of the SPAN elements whose ID begins with "blank."

As the user releases the mouse button, the `onDrop` event handler invokes the `handleDrop()` function. This function retrieves the string data from `event.dataTransfer` and restores it to an array data type (using the `String.split()` method). A little bit of testing makes sure that the word type ("noun" or "adjective") is associated with the desired blank. If so, the source element's text is set to the drop target's `innerText` property; otherwise, an error message is assembled to help the user know what went wrong.

Listing 15-37: **Using Drag-Related Event Handlers**

```
<HTML>
<HEAD>
<TITLE>Dragging Event Handlers</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
TH {text-decoration:underline}
.blanks {text-decoration:underline}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
var timer
function setupDrag() {
    if (event.srcElement.tagName != "TD") {
        // don't allow dragging for any other elements
        event.returnValue = false
    } else {
        // setup array of data to be passed to drop target
        var passedData = [event.srcElement.innerText,
event.srcElement.className]
        // store it as a string
        event.dataTransfer.setData("Text", passedData.join(":"))
        event.dataTransfer.effectAllowed = "copy"
        timer = new Date()
    }
}
function timeIt() {
    if (event.srcElement.tagName == "TD" && timer) {
        if ((new Date()) - timer > 2000) {
            alert("Sorry, time is up. Try again.")
            timer = 0
        }
    }
}
function handleDrop() {
    var elem = event.srcElement
    var passedData = event.dataTransfer.getData("Text")
    var errMsg = ""
    if (passedData) {
        // reconvert passed string to an array
        passedData = passedData.split(":")
        if (elem.id == "blank1") {
            if (passedData[1] == "noun") {
                event.dataTransfer.dropEffect = "copy"
                event.srcElement.innerText = passedData[0]
            } else {
                errMsg = "You can't put an adjective into the noun placeholder."
            }
        } else if (elem.id == "blank2") {
            if (passedData[1] == "adjective") {
                event.dataTransfer.dropEffect = "copy"
```

Continued

Listing 15-37 *(continued)*

```
                event.srcElement.innerText = passedData[0]
            } else {
                errMsg = "You can't put a noun into the adjective placeholder."
            }
        }
        if (errMsg) {
            alert(errMsg)
        }
    }
}
function cancelDefault() {
    if (event.srcElement.id.indexOf("blank") == 0) {
        event.dataTransfer.dropEffect = "copy"
        event.returnValue = false
    }
}
</SCRIPT>
</HEAD>
<BODY onDragStart="setupDrag()" onDrag="timeIt()">
<H1>Dragging Event Handlers</H1>
<HR>
<P>Your goal is to drag one noun and one
adjective from the following table into the blanks
of the sentence. Select a word from the table and
drag it to the desired blank. When you release the
mouse, the word will appear in the blank. You have
two seconds to complete each blank.</P>

<TABLE CELLPADDING=5>
<TR><TH>Nouns</TH><TH>Adjectives</TH></TR>
<TR><TD class="noun">truck</TD><TD class="adjective">round</TD></TR>
<TR><TD class="noun">doll</TD><TD class="adjective">red</TD></TR>
<TR><TD class="noun">ball</TD><TD class="adjective">pretty</TD></TR>
</TABLE>

<P ID="myP" onDragEnter="cancelDefault()" onDragOver="cancelDefault()"
onDrop="handleDrop()">
Pat said, "Oh my, the <SPAN ID="blank1" CLASS="blanks">
     </SPAN>
is so <SPAN ID="blank2" CLASS="blanks">
     </SPAN>!"</P>

<BUTTON onClick="location.reload()">Reset</BUTTON>
</BODY>
</HTML>
```

One event handler not shown in Listing 15-37 is `onDragEnd`. You can use this event to display the elapsed time for each successful drag operation. Because the

event fires on the drag source element, you can implement it in the <BODY> tag and filter events similar to the way the onDragStart or onDrag event handlers filter events for the TD element.

onDragEnter
onDragLeave

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-38 shows the onDragEnter and onDragLeave event handlers in use. The simple page displays (via the status bar) the time of entry to one element of the page. When the dragged cursor leaves the element, the onDragLeave event handler hides the status bar message. No drop target is defined for this page, so when you drag the item, the cursor remains as the "no drop" cursor.

Listing 15-38: Using onDragEnter and onDragLeave Event Handlers

```
<HTML>
<HEAD>
<TITLE>onDragEnter and onDragLeave Event Handlers</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function showEnter() {
    status = "Entered at: " + new Date()
    event.returnValue = false
}
function clearMsg() {
    status = ""
    event.returnValue = false
}
</SCRIPT>
</HEAD>
<BODY>
<H1 onDragEnter="showEnter()" onDragLeave="clearMsg()">
onDragEnter and onDragLeave Event Handlers
</H1>
<HR>
<P>Select any character(s) from this paragraph,
and slowly drag it around the page. When the dragging action enters the
large header above, the status bar displays when the onDragEnter
event handler fires. When you leave the header, the message is cleared
via the onDragLeave event handler.</P>
</BODY>
</HTML>
```

onDragOver

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listing 15-37 of the `onDrag` event handler to see how the `onDragOver` event handler contributes to making an element a drop target.

onDragStart

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 15-37 of the `onDrag` event handler to see how to apply the `onDragStart` event handler in a typical drag-and-drop scenario.

onDrop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listing 15-37 of the `onDrag` event handler to see how to apply the `onDrop` event handler in a typical drag-and-drop scenario.

onFilterChange

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 15-39 demonstrates how the `onFilterChange` event handler can trigger a second transition effect after another one completes. The `onLoad` event handler

triggers the first effect. Although the `onFilterChange` event handler works with most of the same objects in IE4 as IE5, the filter object transition properties are not reflected in a convenient form. The syntax shown in Listing 15-39 uses the new ActiveX filter control found in IE5.5 (described in Chapter 30 of the *JavaScript Bible*).

Listing 15-39: **Using the onFilterChange Event Handler**

```
<HTML>
<HEAD>
<TITLE>onFilterChange Event Handler</TITLE>
<SCRIPT LANGUAGE=JavaScript>
function init() {
    image1.filters[0].apply()
    image2.filters[0].apply()
    start()
}

function start() {
    image1.style.visibility = "hidden"
    image1.filters[0].play()
}

function finish() {
    // verify that first transition is done (optional)
    if (image1.filters[0].status == 0) {
        image2.style.visibility = "visible"
        image2.filters[0].play()
    }
}
</SCRIPT>
</HEAD>
<BODY onLoad="init()">
<H1>onFilterChange Event Handler</H1>
<HR>
<P>The completion of the first transition ("circle-in")
triggers the second ("circle-out").
<BUTTON onClick="location.reload()">Play It Again</BUTTON></P>
<DIV ID="image1" STYLE="visibility:visible;
    position:absolute; top:150px; left:150px;
    filter:progID:DXImageTransform.Microsoft.Iris(irisstyle='CIRCLE',
    motion='in')"
    onFilterChange="finish()"><IMG SRC="desk1.gif" HEIGHT=90
    WIDTH=120></DIV>
<DIV ID="image2" STYLE="visibility:hidden;
    position:absolute; top:150px; left:150px;
    filter:progID:DXImageTransform.Microsoft.Iris(irisstyle='CIRCLE',
    motion='out')">
    <IMG SRC="desk3.gif" HEIGHT=90 WIDTH=120></DIV>
</BODY>
</HTML>
```

onFocus

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listing 15-34 earlier in this chapter for an example of the `onFocus` and `onBlur` event handlers.

onHelp

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 15-40 is a rudimentary example of a context-sensitive help system that displays help messages tailored to the kind of text input required by different text fields. When the user gives focus to either of the text fields, a small legend appears to remind the user that help is available by a press of the F1 help key. IE5/Mac provides only generic help.

Listing 15-40: **Creating Context-Sensitive Help**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function showNameHelp() {
    alert("Enter your first and last names.")
    event.cancelBubble = true
    return false
}
function showYOBHelp() {
    alert("Enter the four-digit year of your birth. For example: 1972")
    event.cancelBubble = true
    return false
}
function showGenericHelp() {
    alert("All fields are required.")
    event.cancelBubble = true
    return false
}
function showLegend() {
    document.all.legend.style.visibility = "visible"/
}
```

```
function hideLegend() {
    document.all.legend.style.visibility = "hidden"
}
function init() {
    var msg = ""
    if (navigator.userAgent.indexOf("Mac") != -1) {
        msg = "Press \'help\' key for help."
    } else if (navigator.userAgent.indexOf("Win") != -1) {
        msg = "Press F1 for help."
    }
    document.all.legend.style.visibility = "hidden"
    document.all.legend.innerHTML = msg
}
</SCRIPT>
</HEAD>

<BODY onLoad="init()" onHelp="return showGenericHelp()">
<H1>onHelp Event Handler</H1>
<HR>
<P ID="legend" STYLE="visibility:hidden; font-size:10px"> </P>
<FORM>
Name: <INPUT TYPE="text" NAME="name" SIZE=30
    onFocus="showLegend()" onBlur="hideLegend()"
    onHelp="return showNameHelp()">
<BR>
Year of Birth: <INPUT TYPE="text" NAME="YOB" SIZE=30
    onFocus="showLegend()" onBlur="hideLegend()"
    onHelp="return showYOBHelp()">
</FORM>
</BODY>
</HTML>
```

onKeyDown
onKeyPress
onKeyUp

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Listing 15-41 is a working laboratory that you can use to better understand the way keyboard event codes and modifier keys work in IE5+ and NN6. The actual code of the listing is less important than watching the page while you use it. For every key or key combination that you press, the page shows the `keyCode` value for the

onKeyDown, onKeyPress, and onKeyUp events. If you hold down one or more modifier keys while performing the key press, the modifier key name is highlighted for each of the three events. Note that when run in NN6, the keyCode value is not the character code (which doesn't show up in this example for NN6). Also, you may need to click the NN6 page for the document object to recognize the keyboard events.

The best way to watch what goes on during keyboard events is to press and hold a key to see the key codes for the onKeyDown and onKeyPress events (see Figure 1-5). Then release the key to see the code for the onKeyUp event. Notice, for instance, that if you press the A key without any modifier key, the onKeyDown event key code is 65 (A) but the onKeyPress key code in IE (and the charCode property in NN6 if it were displayed here) is 97 (a). If you then repeat the exercise but hold the Shift key down, all three events generate the 65 (A) key code (and the Shift modifier labels are highlighted). Releasing the Shift key causes the onKeyUp event to show the key code for the Shift key.

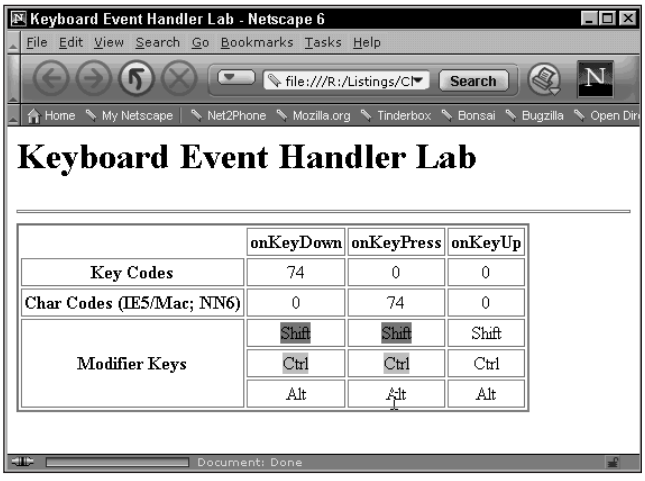


Figure 1-5: Pressing Ctrl+Alt+J in the keyboard event lab page

In another experiment, press any of the four arrow keys. No key code is passed for the onKeyPress event because those keys don't generate those events. They do, however, generate onKeyDown and onKeyUp events.

Listing 15-41: **Keyboard Event Handler Laboratory**

```
<HTML>
<HEAD>
<TITLE>Keyboard Event Handler Lab</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function init() {
    document.onkeydown = showKeyDown
    document.onkeyup = showKeyUp
```

```
    document.onkeypress = showKeyPress
}

function showKeyDown(evt) {
    evt = (evt) ? evt : window.event
    document.getElementById("pressKeyCode").innerHTML = 0
    document.getElementById("upKeyCode").innerHTML = 0
    document.getElementById("pressCharCode").innerHTML = 0
    document.getElementById("upCharCode").innerHTML = 0
    restoreModifiers("")
    restoreModifiers("Down")
    restoreModifiers("Up")
    document.getElementById("downKeyCode").innerHTML = evt.keyCode
    if (evt.charCode) {
        document.getElementById("downCharCode").innerHTML = evt.charCode
    }
    showModifiers("Down", evt)
}
function showKeyUp(evt) {
    evt = (evt) ? evt : window.event
    document.getElementById("upKeyCode").innerHTML = evt.keyCode
    if (evt.charCode) {
        document.getElementById("upCharCode").innerHTML = evt.charCode
    }
    showModifiers("Up", evt)
    return false
}
function showKeyPress(evt) {
    evt = (evt) ? evt : window.event
    document.getElementById("pressKeyCode").innerHTML = evt.keyCode
    if (evt.charCode) {
        document.getElementById("pressCharCode").innerHTML = evt.charCode
    }
    showModifiers("", evt)
    return false
}
function showModifiers(ext, evt) {
    restoreModifiers(ext)
    if (evt.shiftKey) {
        document.getElementById("shift" + ext).style.backgroundColor = "#ff0000"
    }
    if (evt.ctrlKey) {
        document.getElementById("ctrl" + ext).style.backgroundColor = "#00ff00"
    }
    if (evt.altKey) {
        document.getElementById("alt" + ext).style.backgroundColor = "#0000ff"
    }
}
```

Continued

Listing 15-41 *(continued)*

```
function restoreModifiers(ext) {
    document.getElementById("shift" + ext).style.backgroundColor = "#ffffff"
    document.getElementById("ctrl" + ext).style.backgroundColor = "#ffffff"
    document.getElementById("alt" + ext).style.backgroundColor = "#ffffff"
}
</SCRIPT>
</HEAD>

<BODY onLoad="init()">
<H1>Keyboard Event Handler Lab</H1>
<HR>
<FORM>
<TABLE BORDER=2 CELLPADDING=2>
<TR><TH></TH><TH>onKeyDown</TH><TH>onKeyPress</TH><TH>onKeyUp</TH></TR>
<TR><TH>Key Codes</TH>
    <TD ID="downKeyCode">0</TD>
    <TD ID="pressKeyCode">0</TD>
    <TD ID="upKeyCode">0</TD>
</TR>
<TR><TH>Char Codes (IE5/Mac; NN6)</TH>
    <TD ID="downCharCode">0</TD>
    <TD ID="pressCharCode">0</TD>
    <TD ID="upCharCode">0</TD>
</TR>
<TR><TH ROWSPAN=3>Modifier Keys</TH>
    <TD><SPAN ID="shiftDown">Shift</SPAN></TD>
    <TD><SPAN ID="shift">Shift</SPAN></TD>
    <TD><SPAN ID="shiftUp">Shift</SPAN></TD>
</TR>
<TR>
    <TD><SPAN ID="ctrlDown">Ctrl</SPAN></TD>
    <TD><SPAN ID="ctrl">Ctrl</SPAN></TD>
    <TD><SPAN ID="ctrlUp">Ctrl</SPAN></TD>
</TR>
<TR>
    <TD><SPAN ID="altDown">Alt</SPAN></TD>
    <TD><SPAN ID="alt">Alt</SPAN></TD>
    <TD><SPAN ID="altUp">Alt</SPAN></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Spend some time with this lab, and try all kinds of keys and key combinations until you understand the way the events and key codes work.

onLoseCapture

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listing 15-30 earlier in this chapter for an example of how to use `onLoseCapture` with an event-capturing scenario for displaying a context menu. The `onLoseCapture` event handler hides the context menu when the user performs any action that causes the menu to lose mouse capture.

onMouseDown
onMouseUp

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

To demonstrate a likely scenario of changing button images in response to rolling atop an image, pressing down on it, releasing the mouse button, and rolling away from the image, Listing 15-42 presents a pair of small navigation buttons (left- and right-arrow buttons). Because the image object is not part of the document object model for NN2 or IE3 (which reports itself as Navigator version 2), the page is designed to accept all browsers. Only those browsers that support precached images and image swapping (and thus pass the test for the presence of the `document.images` array) can execute those statements. For a browser with an image object, images are preloaded into the browser cache as the page loads so that response to the user is instantaneous the first time the user calls upon new versions of the images.

Listing 15-42: Using onMouseDown and onMouseUp Event Handlers

```
<HTML>
<HEAD>
<TITLE>onMouseDown and onMouseUp Event Handlers</TITLE>
<SCRIPT LANGUAGE="JavaScript">
if (document.images) {
    var RightNormImg = new Image(16,16)
    var RightUpImg = new Image(16,16)
```

Continued

Listing 15-42 *(continued)*

```
    var RightDownImg = new Image(16,16)
    var LeftNormImg = new Image(16,16)
    var LeftUpImg = new Image(16,16)
    var LeftDownImg = new Image(16,16)

    RightNormImg.src = "RightNorm.gif"
    RightUpImg.src = "RightUp.gif"
    RightDownImg.src = "RightDown.gif"
    LeftNormImg.src = "LeftNorm.gif"
    LeftUpImg.src = "LeftUp.gif"
    LeftDownImg.src = "LeftDown.gif"
}
function setImage(imgName, type) {
    if (document.images) {
        var imgFile = eval(imgName + type + "Img.src")
        document.images[imgName].src = imgFile
        return false
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>onMouseDown and onMouseUp Event Handlers</H1>
<HR>
<P>Roll atop and click on the buttons to see how the link event handlers swap
images:</P>
<CENTER>
<A HREF="javascript:void(0)"
    onMouseOver="return setImage('Left','Up')"
    onMouseDown="return setImage('Left','Down')"
    onMouseUp="return setImage('Left','Up')"
    onMouseOut="return setImage('Left','Norm')"
>
<IMG NAME="Left" SRC="LeftNorm.gif" HEIGHT=16 WIDTH=16 BORDER=0></A>
  
<A HREF="javascript:void(0)"
    onMouseOver="return setImage('Right','Up')"
    onMouseDown="return setImage('Right','Down')"
    onMouseUp="return setImage('Right','Up')"
    onMouseOut="return setImage('Right','Norm')"
>
<IMG NAME="Right" SRC="RightNorm.gif" HEIGHT=16 WIDTH=16 BORDER=0></A>
</CENTER>
</BODY>
</HTML>
```

IE4+ and NN6+ simplify the implementation of this kind of three-state image button by allowing you to assign the event handlers directly to IMG element objects. Wrapping images inside links is a backward compatibility approach that allows older browsers to respond to clicks on images for navigation or other scripting tasks.

onMouseEnter
onMouseLeave

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

You can modify Listing 15-43 with the IE5.5 syntax by substituting `onMouseEnter` for `onMouseOver` and `onMouseLeave` for `onMouseOut`. The effect is the same.

onMouseMove

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			(✓)	✓			✓	✓	✓

Example

Listing 15-43 is a simplified example of dragging elements in IE4+. (See Chapter 31 of the *JavaScript Bible* for more dragging examples.) Three images are individually positioned on the page. Most of the scripting code concerns itself with the geography of click locations, the stacking order of the images, and the management of the `onMouseMove` event handler so that it is active only when an item is dragged.

Scripts assign the `onMouseDown` and `onMouseUp` event handlers to the `document` object, invoking the `engage()` and `release()` functions, respectively. When a user mouses down anywhere in the document, the `engage()` function starts by invoking `setSelectedObj()`. This function examines the target of the `mouseDown` event. If it is one of the map images, the `selectedObj` global variable is set to the `image` object and the element is brought to the front of the stacking order of images (any previously stacked image is returned to its normal position in the stack). `MouseDown` events on any other element simply make sure that the `selectedObj` variable is `null`. The presence of a value assigned to `selectedObj` serves as a kind of switch for other functions: When the variable contains a value, it means that the user is doing something associated with dragging an element.

Back at the `engage()` function — provided the user mouses down on one of the draggable images — the `onMouseMove` event handler is assigned to the `document` object, setting it to invoke the `dragIt()` function. For the sake of users, the offset of the mouse down event from the top-left corner of the image is preserved in the

offsetX and offsetY variables (minus any scrolling that the body is subject to at that instant). These offset values are necessary to let the scripts set the location of the image during dragging (the location is set for the top-left corner of the image) while keeping the cursor in the same location within the image as when the user first presses the mouse.

As the user drags the image, the onMouseDown event handler fires repeatedly, allowing the dragIt() function to continually update the location of the element relative to the current cursor position (the event.clientX and event.clientY properties). The global offset variables are subtracted from the cursor position to preserve the relation of the image's top-left corner to the initial cursor position at mouse down.

Upon the user releasing the mouse button, the release() function turns off the onMouseMove event handler (setting it to null). This prevents the event from being processed at all during normal usage of the page. The selectedObj global variable is also set to null, turning off the "switch" that indicates dragging is in session.

Listing 15-43: **Dragging Elements with onMouseMove**

```
<HTML>
<HEAD><TITLE>onMouseMove Event Handler</TITLE>
<STYLE TYPE="text/css">
    #camap {position:absolute; left:20; top:120}
    #ormap {position:absolute; left:80; top:120}
    #wamap {position:absolute; left:140; top:120}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
// global variables used while dragging
var offsetX = 0
var offsetY = 0
var selectedObj
var frontObj

// set document-level event handlers
document.onmousedown = engage
document.onmouseup = release

// positioning an object at a specific pixel coordinate
function shiftTo(obj, x, y) {
    obj.style.pixelLeft = x
    obj.style.pixelTop = y
}

// setting the z-order of an object
function bringToFront(obj) {
    if (frontObj) {
        frontObj.style.zIndex = 0
    }
    frontObj = obj
    frontObj.style.zIndex = 1
}
```

```
// set global var to a reference to dragged element
function setSelectedObj() {
    var imgObj = window.event.srcElement
    if (imgObj.id.indexOf("map") == 2) {
        selectedObj = imgObj
        bringToFront(selectedObj)
        return
    }
    selectedObj = null
    return
}

// do the dragging (called repeatedly by onMouseMove)
function dragIt() {
    if (selectedObj) {
        shiftTo(selectedObj, (event.clientX - offsetX), (event.clientY -
offsetY))
        return false
    }
}

// set global vars and turn on mousemove trapping (called by onMouseDown)
function engage() {
    setSelectedObj()
    if (selectedObj) {
        document.onmousemove = dragIt
        offsetX = window.event.offsetX - document.body.scrollLeft
        offsetY = window.event.offsetY - document.body.scrollTop
    }
}

// restore everything as before (called by onMouseUp)
function release() {
    if (selectedObj) {
        document.onmousemove = null
        selectedObj = null
    }
}

</SCRIPT>
</HEAD>
<BODY>
<H1>onMouseMove Event Handler</H1>
<HR>
Click and drag the images:
<IMG ID="camap" SRC="camap.gif" WIDTH="47" HEIGHT="82" BORDER="0">
<IMG ID="ormap" SRC="ormap.gif" WIDTH="57" HEIGHT="45" BORDER="0">
<IMG ID="wamap" SRC="wamap.gif" WIDTH="38" HEIGHT="29" BORDER="0">
</SCRIPT>
</BODY>
</HTML>
```

onMouseOut
onMouseOver

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 15-44 uses the U.S. Pledge of Allegiance with four links to demonstrate how to use the `onMouseOver` and `onMouseOut` event handlers. Notice that for each link, the handler runs a general-purpose function that sets the window's status message. The function returns a `true` value, which the event handler call evaluates to replicate the required `return true` statement needed for setting the status bar. In one status message, I supply a URL in parentheses to let you evaluate how helpful you think it is for users.

Listing 15-44: Using onMouseOver and onMouseOut Event Handlers

```
<HTML>
<HEAD>
<TITLE>onMouseOver and onMouseOut Event Handlers</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setStatus(msg) {
    status = msg
    return true
}
// destination of all link HREFs
function emulate() {
    alert("Not going there in this demo.")
}
</SCRIPT>
</HEAD>
<BODY>
<H1>onMouseOver and onMouseOut Event Handlers
</H1>
<HR>
<H1>Pledge of Allegiance</H1>
<HR>
```

```
I pledge <A HREF="javascript:emulate()" onMouseOver="return setStatus('View
dictionary definition')" onMouseOut="return setStatus('')">allegiance</A> to the
<A HREF="javascript:emulate()" onMouseOver="return setStatus('Learn about the
U.S. flag (http://lcweb.loc.gov)')" onMouseOut="return setStatus('')">flag</A>
of the <A HREF="javascript:emulate()" onMouseOver="return setStatus('View info
about the U.S. government')" onMouseOut="return setStatus('')">United States of
America</A>, and to the Republic for which it stands, one nation <A
HREF="javascript:emulate()" onMouseOver="return setStatus('Read about the
history of this phrase in the Pledge')" onMouseOut="return setStatus('')">under
God</A>, indivisible, with liberty and justice for all.
</BODY>
</HTML>
```

onPaste

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 15-45 demonstrates how to use the `onBeforePaste` and `onPaste` event handlers (in conjunction with `onBeforeCopy` and `onCopy`) to let scripts control the data transfer process during a copy-and-paste user operation. A table contains words to be copied (one column of nouns, one column of adjectives) and then pasted into blanks in a paragraph. The `onBeforeCopy` and `onCopy` event handlers are assigned to the TABLE element because the events from the TD elements bubble up to the TABLE container and there is less HTML code to contend with.

Inside the paragraph, two SPAN elements contain underscored blanks. To paste text into the blanks, the user must first select at least one character of the blanks. (See Listing 15-37, which gives a drag-and-drop version of this application.) The `onBeforePaste` event handler in the paragraph (which gets the event as it bubbles up from either SPAN) sets the `event.returnValue` property to `false`, thus allowing the Paste item to appear in the context and Edit menus (not a normal occurrence in HTML body content).

At paste time, the `innerHTML` property of the target SPAN is set to the text data stored in the clipboard. The `event.returnValue` property is set to `false` here, as well, to prevent normal system pasting from interfering with the controlled version.

Listing 15-45: Using onBeforePaste and onPaste Event Handlers

```
<HTML>
<HEAD>
<TITLE>onBeforePaste and onPaste Event Handlers</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
TH {text-decoration:underline}
.blanks {text-decoration:underline}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function selectWhole() {
    var obj = window.event.srcElement
    var range = document.body.createTextRange()
    range.moveToElementText(obj)
    range.select()
    event.returnValue = false
}
function handleCopy() {
    var rng = document.selection.createRange()
    clipboardData.setData("Text",rng.text)
    event.returnValue = false
}

function handlePaste() {
    var elem = window.event.srcElement
    if (elem.className == "blanks") {
        elem.innerHTML = clipboardData.getData("Text")
    }
    event.returnValue = false
}
function handleBeforePaste() {
    var elem = window.event.srcElement
    if (elem.className == "blanks") {
        event.returnValue = false
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>onBeforePaste and onPaste Event Handlers</H1>
<HR>
<P>Your goal is to copy and paste one noun and one
adjective from the following table into the blanks
of the sentence. Select a word from the table and
copy it to the clipboard. Select one or more spaces
of the blanks in the sentence and choose Paste to
replace the blank with the clipboard contents.</P>

<TABLE CELLPADDING=5 onBeforeCopy="selectWhole()" onCopy="handleCopy()" >
<TR><TH>Nouns</TH><TH>Adjectives</TH></TR>
```

```
<TR><TD>truck</TD><TD>round</TD></TR>
<TR><TD>doll</TD><TD>red</TD></TR>
<TR><TD>ball</TD><TD>pretty</TD></TR>
</TABLE>

<P ID="myP" onBeforePaste="handleBeforePaste()" onPaste="handlePaste()">
Pat said, "Oh my, the <SPAN ID="blank1" CLASS="blanks">
     </SPAN>
is so <SPAN ID="blank2" CLASS="blanks">
     </SPAN>!"</P>

<BUTTON onClick="location.reload()">Reset</BUTTON>
</BODY>
</HTML>
```

onPropertyChange

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

The page generated by Listing 15-46 contains four radio buttons that alter the `innerHTML` and `style.color` properties of a paragraph. The paragraph's `onPropertyChange` event handler invokes the `showChange()` function, which extracts information about the event and displays the data in the status bar of the window. Notice how the property name includes `style.` when you modify the style sheet property.

Listing 15-46: **Using the onPropertyChange Property**

```
<HTML>
<HEAD>
<TITLE>onPropertyChange Event Handler</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function normalText() {
    myP.innerText = "This is a sample paragraph."
}
function shortText() {
    myP.innerText = "Short stuff."
}
function normalColor() {
    myP.style.color = "black"
}
```

Continued

Listing 15-46 *(continued)*

```
function hotColor() {
    myP.style.color = "red"
}
function showChange() {
    var objID = event.srcElement.id
    var propName = event.propertyName
    var newValue = eval(objID + "." + propName)
    status = "The " + propName + " property of the " + objID
    status += " object has changed to \"" + newValue + "\"."
}
</SCRIPT>
</HEAD>
<BODY>
<H1>onPropertyChange Event Handler</H1>
<HR>
<P ID="myP" onPropertyChange = "showChange()">This is a sample paragraph.</P>
<FORM>
Text: <INPUT TYPE="radio" NAME="btn1" CHECKED onClick="normalText()">Normal
      <INPUT TYPE="radio" NAME="btn1" onClick="shortText()">Short
<BR>
Color: <INPUT TYPE="radio" NAME="btn2" CHECKED onClick="normalColor()">Black
       <INPUT TYPE="radio" NAME="btn2" onClick="hotColor()">Red
</FORM>
</BODY>
</HTML>
```

onReadyStateChange

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

You can use the `onReadyStateChange` event handler to assist with a status display while a long external file, such as a Java applet, loads. For example, you might have a small image on a page that changes with the state change of an applet. The `<APPLET>` tag assigns a function to the `onReadyStateChange` event handler:

```
<APPLET ... onReadyStateChange="showState(this)">
```

Then the function changes the image for each state type:

```
function showState(obj) {
    var img = document.all.statusImage
    switch (obj.readyState) {
        case "uninitialized" :
            img.src = uninit.src
            break
        case "loading" :
            img.src = loading.src
            break
        case "complete" :
            img.src = ready.src
    }
}
```

The preceding function assumes that the state images are precached as the page loads.

onResize

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

If you want to capture the user's resizing of the browser window (or frame), you can assign a function to the `onResize` event handler either via script

```
window.onresize = handleResize
```

or by an HTML attribute of the BODY element:

```
<BODY onResize="handleResize()">
```

onSelectStart

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use the page from Listing 15-47 to see how the `onSelectStart` event handler works when a user selects across multiple elements on a page. As the user begins a selection anywhere on the page, the ID of the object receiving the event appears in the status bar. Notice that the event doesn't fire until you actually make a selection. When no other element is under the cursor, the BODY element fires the event.

Listing 15-47: **Using the onSelectStart Event Handler**

```
<HTML>
<HEAD>
<TITLE>onSelectStart Event Handler</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function showObj() {
    var objID = event.srcElement.id
    status = "Selection started with object: " + objID
}
</SCRIPT>
</HEAD>
<BODY ID="myBody" onSelectStart="showObj()">
<H1 ID="myH1">onSelectStart Event Handler</H1>
<HR ID="myHR">
<P ID="myP">This is a sample paragraph.</P>
<TABLE BORDER="1">
<TR ID="row1">
    <TH ID="header1">Column A</TH>
    <TH ID="header2">Column B</TH>
    <TH ID="header3">Column C</TH>
</TR>
<TR ID="row2">
    <TD ID="cellA2">text</TD>
    <TD ID="cellB2">text</TD>
    <TD ID="cellC2">text</TD>
</TR>
<TR ID="row3">
    <TD ID="cellA3">text</TD>
    <TD ID="cellB3">text</TD>
    <TD ID="cellC3">text</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

CHAPTER 2

Window and Frame Objects (Chapter 16)

In This Chapter

Scripting communication among multiple frames

Creating and managing new windows

Controlling the size, position, and appearance of the browser window

Dynamically adjusting frame sizes and frameset compositions

As physical containers of documents, window and frame objects play huge rolls in scripting. The `window` object has been scriptable in one form or another since the first scriptable browsers. Of course the object has gained numerous properties, methods, and event handlers over time, but you also often find many object-model-specific items that you probably wish were available across all browsers.

While scripts permit Web authors to manage multiple windows — and many of the examples in this chapter support that facility — try to think about your visitors, too. Very often multiple windows get in the way of site navigation and content, regardless of your good intentions. As some examples also demonstrate, you must include safety nets for your code to counteract the unpredictable actions of users who close or hide windows precisely when you don't want them to do so. Therefore, do not regard the multi-window examples here as user interface recommendations; rather consider them as recommended ways to handle a potentially tricky user-interface element.

Possible exceptions to my multi-window admonitions are the modal and modeless dialog box windows provided by various versions of IE for Windows. For other platforms, a modal dialog box can be simulated (search for details at `www.dannyg.com`). IE5.5 for Windows also adds a popup type window, which can be a helpful user interface element that exists between a tooltip and a modal dialog box.

Modern browsers, however, provide ample script control over framesets. As examples in this chapter demonstrate, your scripts can hide and show frames, or completely rearchitect a frameset without loading a new frameset.

Examples Highlights

- Listing 16-4 for the `window.closed` property demonstrates an industrial-strength treatment of new window creation, which works with all scriptable browsers (taking into account shortcomings of earlier browsers).
- NN4+ allows dynamic control over the presence of window chrome (statusbar, toolbar, et al.) with the help of signed scripts, as shown in Listing 16-6. Without signed scripts, or for IE, you must use `window.open()` to create a separate window with the characteristics of your choice.
- The example listings for the `window.opener` property show you how scripts from a subwindow communicate with the window that opened it.
- In the example listings for the `window.parent` property, you see how references to the various synonyms for a window object within a frameset evaluate. Thus, you can see what the references `window`, `top`, `parent`, and `self` mean within a frameset.
- Compare Listings 16-20, 16-23, and 16-29 to understand not only the different looks of the three native dialog box windows (alert, confirm, and prompt), but also how values returned from two of them can influence script processing sequences.
- A simple countdown timer in Listing 16-22 shows a practical application of the `window.clearTimeout()` method. Here the method stops the looping timer when the count reaches zero.
- Watch the browser window dance in Listing 16-24. The `window.moveBy()` and `window.moveTo()` methods put window positioning through its paces.
- Examples for `window.setInterval()` and `window.setTimeout()` apply these two similar methods to applications that are ideal for each one. You find other applications of `setTimeout()` in examples for the `window.closed` property and `window.open()` method.
- Internet Explorer's modal and modeless dialog box windows get workouts in Listings 16-39 through 16-42.
- The composition of a frameset, including the sizes of the frames, can be controlled dynamically in IE4+ and NN6, as shown in examples for the `FRAMESET.cols` and `FRAMESET.rows` properties.

Window Object

Properties

clipboardData

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listings 15-30 and 15-39 (in Chapter 1 of this book) to see how the `clipboardData` object is used with a variety of edit-related event handlers.

closed

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

In Listing 16-4, I have created the ultimate cross-platform window opening and closing sample. It takes into account the lack of the `opener` property in Navigator 2, the missing `closed` property in Navigator 2 and Internet Explorer 3, and it even provides an ugly but necessary workaround for the inability of Internet Explorer 3 to gracefully see if a subwindow is still open.

The script begins by initializing a global variable, `newWind`, which is used to hold the object reference to the second window. This value needs to be global so that other functions can reference the window for tasks, such as closing. Another global variable, `isIE3`, is a Boolean flag that lets the window closing routines know whether the visitor is using Internet Explorer 3 (see details about the `navigator.appVersion` property in Chapter 28 of the *JavaScript Bible*).

For this example, the new window contains some HTML code written dynamically to it, rather than loading an existing HTML file into it. Therefore, the URL parameter of the `window.open()` method is left as an empty string. It is vital, however, to assign a name in the second parameter to accommodate the Internet Explorer 3 workaround for closing the window. After the new window is opened, an `opener` property is assigned to the object if one is not already assigned (this property is needed only for Navigator 2). Next comes a brief delay to allow Internet Explorer (especially versions 3 and 4) to catch up with opening the window so that content

can be written to it. The delay (using the `setTimeout()` method described later in this chapter) invokes the `finishNewWindow()` function, which uses the global `newWind` variable to reference the window for writing. The `document.close()` method closes writing to the document — a different kind of close than a window close.

A separate function, `closeWindow()`, is responsible for closing the subwindow. To accommodate Internet Explorer 3, the script appears to create another window with the same characteristics as the one opened earlier in the script. This is the trick: If the earlier window exists (with exactly the same parameters and a name *other* than an empty string), Internet Explorer does not create a new window even with the `window.open()` method executing in plain sight. To the user, nothing unusual appears on the screen. Things look weird for Internet Explorer 3 users only if the user has closed the subwindow. The `window.open()` method momentarily creates that subwindow. This subwindow is necessary because a "living" `window` object must be available for the upcoming test of window existence. (Internet Explorer 3 displays a script error if you try to address a missing window, while NN2+ and IE4+ simply return friendly `null` values.)

As a final test, an `if` condition looks at two conditions: 1) if the `window` object has ever been initialized with a value other than `null` (in case you click the window closing button before ever having created the new window) and 2) if the window's `closed` property is `null` or `false`. If either condition is true, the `close()` method is sent to the second window.

Listing 16-4: **Checking Before Closing a Window**

```
<HTML>
<HEAD>
<TITLE>window.closed Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// initialize global var for new window object
// so it can be accessed by all functions on the page
var newWind
// set flag to help out with special handling for window closing
var isIE3 = (navigator.appVersion.indexOf("MSIE 3") != -1) ? true : false
// make the new window and put some stuff in it
function newWindow() {
    newWind = window.open("","subwindow","HEIGHT=200,WIDTH=200")
    // take care of Navigator 2
    if (newWind.opener == null) {
        newWind.opener = window
    }
    setTimeout("finishNewWindow()", 100)
}
function finishNewWindow() {
    var output = ""
    output += "<HTML><BODY><H1>A Sub-window</H1>"
    output += "<FORM><INPUT TYPE='button' VALUE='Close Main Window'"
    output +="onClick='window.opener.close()'></FORM></BODY></HTML>"
```

```
    newWind.document.write(output)
    newWind.document.close()
}
// close subwindow, including ugly workaround for IE3
function closeWindow() {
    if (isIE3) {
        // if window is already open, nothing appears to happen
        // but if not, the subwindow flashes momentarily (yech!)
        newWind = window.open("","subwindow","HEIGHT=200,WIDTH=200")
    }
    if (newWind && !newWind.closed) {
        newWind.close()
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Open Window" onClick="newWindow()"><BR>
<INPUT TYPE="button" VALUE="Close it if Still Open" onClick="closeWindow()">
</FORM>
</BODY>
</HTML>
```

To complete the example of the window opening and closing, notice that the subwindow is given a button whose `onClick` event handler closes the main window. In Navigator 2 and Internet Explorer 3, this occurs without complaint. But in NN3+ and IE4+, the user is presented with an alert asking to confirm the closure of the main browser window.

defaultStatus

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Unless you plan to change the default statusbar text while a user spends time at your Web page, the best time to set the property is when the document loads. In Listing 16-5, notice how I also read this property to reset the statusbar in an `onMouseOut` event handler. Setting the `status` property to empty also resets the statusbar to the `defaultStatus` setting.

Listing 16-5: **Setting the Default Status Message**

```
<HTML>
<HEAD>
<TITLE>window.defaultStatus property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
window.defaultStatus = "Welcome to my Web site."
</SCRIPT>
</HEAD>
<BODY>
<A HREF="http://www.microsoft.com"
onMouseOver="window.status = 'Visit Microsoft\'s Home page.';return true"
onMouseOut="window.status = '';return true">Microsoft</A><P>
<A HREF="http://home.netscape.com"
onMouseOver="window.status = 'Visit Netscape\'s Home page.';return true"
onMouseOut="window.status = window.defaultStatus;return true">Netscape</A>
</BODY>
</HTML>
```

If you need to display single or double quotes in the statusbar (as in the second link in Listing 16-5), use escape characters (`\'` and `\"`) as part of the strings being assigned to these properties.

dialogArguments

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 16-38 for the `window.showModalDialog()` method to see how arguments can be passed to a dialog box and retrieved via the `dialogArguments` property.

dialogHeight
dialogWidth

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Dialog boxes sometimes provide a button or icon that reveals more details or more complex settings for advanced users. You can create a function that handles the toggle between two sizes. The following function assumes that the document in the dialog box has a button whose label also toggles between "Show Details" and "Hide Details." The button's `onClick` event handler invokes the function as `toggleDetails(this)`.

```
function toggleDetails(btn) {
    if (dialogHeight == "200px") {
        dialogHeight = "350px"
        btn.value = "Hide Details"
    } else {
        dialogHeight = "200px"
        btn.value = "Show Details"
    }
}
```

In practice, you also have to toggle the `display` style sheet property of the extra material between `none` and `block` to make sure that the dialog box does not display scrollbars in the smaller dialog box version.

dialogLeft
dialogTop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Although usually not a good idea because of the potentially jarring effect on a user, you can reposition a dialog box window that has been resized by script (or by the user if you let the dialog box be resizable). The following statements in a dialog box window document's script recenter the dialog box window.

```
dialogLeft = (screen.availWidth/2) - (parseInt(dialogWidth)/2) + "px"
dialogHeight = (screen.availHeight/2) - (parseInt(dialogHeight)/2) + "px"
```

Note that the `parseInt()` functions are used to read the numeric portion of the `dialogWidth` and `dialogHeight` properties so that the values can be used for arithmetic.

directories
locationbar
menubar
personalbar
scrollbars
statusbar
toolbar

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

In Listing 16-6, you can experiment with the look of a browser window with any of the chrome elements turned on and off. To run this script, you must either sign the scripts or turn on codebase principals (see Chapter 46 of the *JavaScript Bible*). Java must also be enabled to use the signed script statements.

As the page loads, it stores the current state of each chrome element. One button for each chrome element triggers the `toggleBar()` function. This function inverts the visible property for the chrome object passed as a parameter to the function. Finally, the Restore button returns visibility to their original settings. Notice that the `restore()` function is also called by the `onUnload` event handler for the document. Also, if you load this example into NN6, non-fatal script errors occur when the scrollbars are turned on or off.

Listing 16-6: **Controlling Window Chrome**

```
<HTML>
<HEAD>
<TITLE>Bars Bars Bars</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// store original outer dimensions as page loads
var originalLocationbar = window.locationbar.visible
var originalMenubar = window.menubar.visible
var originalPersonalbar = window.personalbar.visible
var originalScrollbars = window.scrollbars.visible
var originalStatusbar = window.statusbar.visible
var originalToolbar = window.toolbar.visible

// generic function to set inner dimensions
function toggleBar(bar) {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite")
    bar.visible = !bar.visible
    netscape.security.PrivilegeManager.revertPrivilege("UniversalBrowserWrite")
}
```

```
// restore settings
function restore() {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite")
    window.locationbar.visible = originalLocationbar
    window.menubar.visible = originalMenubar
    window.personalbar.visible = originalPersonalbar
    window.scrollbars.visible = originalScrollbars
    window.statusbar.visible = originalStatusbar
    window.toolbar.visible = originalToolbar
    netscape.security.PrivilegeManager.revertPrivilege("UniversalBrowserWrite")
}
</SCRIPT>
</HEAD>
<BODY onUnload="restore()">
<FORM>
<B>Toggle Window Bars</B><BR>
<INPUT TYPE="button" VALUE="Location Bar"
onClick="toggleBar(window.locationbar)"><BR>
<INPUT TYPE="button" VALUE="Menu Bar" onClick="toggleBar(window.menubar)"><BR>
<INPUT TYPE="button" VALUE="Personal Bar"
onClick="toggleBar(window.personalbar)"><BR>
<INPUT TYPE="button" VALUE="Scrollbars"
onClick="toggleBar(window.scrollbars)"><BR>
<INPUT TYPE="button" VALUE="Status Bar"
onClick="toggleBar(window.statusbar)"><BR>
<INPUT TYPE="button" VALUE="Tool Bar" onClick="toggleBar(window.toolbar)"><BR>
<HR>
<INPUT TYPE="button" VALUE="Restore Original Settings" onClick="restore()"><BR>
</FORM>
</BODY>
</HTML>
```

external

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The first example asks the user if it is okay to add a Web site to the Active Desktop. If Active Desktop is not enabled, the user is given the choice of enabling it at this point.

```
external.AddDesktopComponent("http://www.nytimes.com","website", 200, 100, 400, 400)
```

In the next example, the user is asked to approve the addition of a URL to the Favorites list. The user can follow the normal procedure for filing the item in a folder in the list.

```
external.AddFavorite("http://www.dannyg.com/update6.html",
"JSBible 4 Support Center")
```

The final example assumes that a user makes a choice from a SELECT list of items. The `onChange` event handler of the SELECT list invokes the following function to navigate to a fictitious page and locate listings for a chosen sports team on the page.

```
function locate(list) {
    var choice = list.options[list.selectedIndex].value
    external.NavigateAndFind("http://www.collegesports.net/scores.html", choice,
"scores")
}
```

frames

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listings 16-7 and 16-8 demonstrate how JavaScript treats values of frame references from objects inside a frame. The same document is loaded into each frame. A script in that document extracts info about the current frame and the entire frameset. Figure 2-1 shows the results after loading the HTML document in Listing 16-7.

Listing 16-7: Framesetting Document for Listing 16-8

```
<HTML>
<HEAD>
<TITLE>window.frames property</TITLE>
</HEAD>
<FRAMESET COLS="50%,50%">
    <FRAME NAME="JustAKid1" SRC="lst16-08.htm">
    <FRAME NAME="JustAKid2" SRC="lst16-08.htm">
</FRAMESET>
</HTML>
```

A call to determine the number (length) of frames returns 0 from the point of view of the current frame referenced. That's because each frame here is a window that has no nested frames within it. But add the `parent` property to the reference, and the scope zooms out to take into account all frames generated by the parent window's document.

Listing 16-8: **Showing Various Window Properties**

```
<HTML>
<HEAD>
<TITLE>Window Revealer II</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function gatherWindowData() {
    var msg = ""
    msg += "<B>From the point of view of this frame:</B><BR>"
    msg += "window.frames.length: " + window.frames.length + "<BR>"
    msg += "window.name: " + window.name + "<P>"
    msg += "<B>From the point of view of the framesetting document:</B><BR>"
    msg += "parent.frames.length: " + parent.frames.length + "<BR>"
    msg += "parent.frames[0].name: " + parent.frames[0].name
    return msg
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write(gatherWindowData())
</SCRIPT>
</BODY>
</HTML>
```

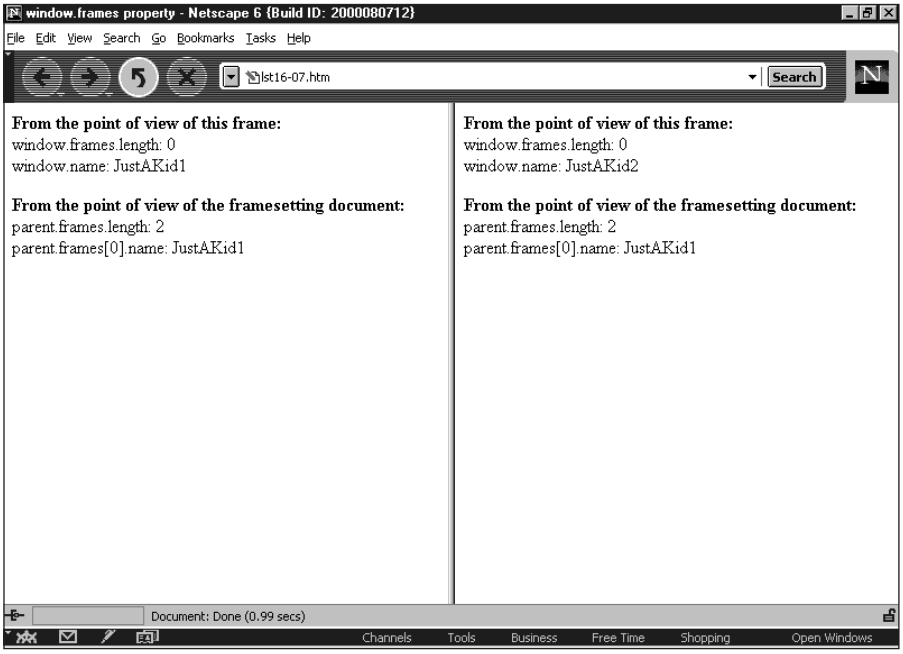


Figure 2-1: Property readouts from both frames loaded from Listing 16-7

The last statement in the example shows how to use the array syntax (brackets) to refer to a specific frame. All array indexes start with 0 for the first entry. Because the document asks for the name of the first frame (`parent.frames[0]`), the response is `JustAKid1` for both frames.

```
innerHeight
innerWidth
outerHeight
outerWidth
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

In Listing 16-9, a number of buttons let you see the results of setting the `innerHeight`, `innerWidth`, `outerHeight`, and `outerWidth` properties.

Listing 16-9: Setting Window Height and Width

```
<HTML>
<HEAD>
<TITLE>Window Sizer</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// store original outer dimensions as page loads
var originalWidth = window.outerWidth
var originalHeight = window.outerHeight
// generic function to set inner dimensions
function setInner(width, height) {
    window.innerWidth = width
    window.innerHeight = height
}
// generic function to set outer dimensions
function setOuter(width, height) {
    window.outerWidth = width
    window.outerHeight = height
}
// restore window to original dimensions
function restore() {
    window.outerWidth = originalWidth
    window.outerHeight = originalHeight
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<B>Setting Inner Sizes</B><BR>
```

```
<INPUT TYPE="button" VALUE="600 Pixels Square" onClick="setInner(600,600)"><BR>
<INPUT TYPE="button" VALUE="300 Pixels Square" onClick="setInner(300,300)"><BR>
<INPUT TYPE="button" VALUE="Available Screen Space"
onClick="setInner(screen.availWidth, screen.availHeight)"><BR>
<HR>
<B>Setting Outer Sizes</B><BR>
<INPUT TYPE="button" VALUE="600 Pixels Square" onClick="setOuter(600,600)"><BR>
<INPUT TYPE="button" VALUE="300 Pixels Square" onClick="setOuter(300,300)"><BR>
<INPUT TYPE="button" VALUE="Available Screen Space"
onClick="setOuter(screen.availWidth, screen.availHeight)"><BR>
<HR>
<INPUT TYPE="button" VALUE="Cinch up for Win95" onClick="setInner(273,304)"><BR>
<INPUT TYPE="button" VALUE="Cinch up for Mac" onClick="setInner(273,304)"><BR>
<INPUT TYPE="button" VALUE="Restore Original" onClick="restore()"><BR>
</FORM>
</BODY>
</HTML>
```

As the document loads, it saves the current outer dimensions in global variables. One of the buttons restores the windows to these settings. Two parallel sets of buttons set the inner and outer dimensions to the same pixel values so that you can see the effects on the overall window and document area when a script changes the various properties.

Because Navigator 4 displays different-looking buttons in different platforms (as well as other elements), the two buttons contain script instructions to size the window to best display the window contents. Unfortunately, no measure of the active area of a document is available, so that the dimension values were determined by trial and error before being hard-wired into the script.

navigator

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

This book is littered with examples of using the `navigator` object, primarily for performing browser detection. Examples of specific `navigator` object properties can be found in Chapter 28 of the *JavaScript Bible* and Chapter 12 of this book.

offscreenBuffering

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

If you want to turn off buffering for an entire page, include the following statement at the beginning of your script statements:

```
window.offscreenBuffering = false
```

onerror

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

In Listing 16-10, one button triggers a script that contains an error. I've added an error-handling function to process the error so that it opens a separate window and fills in a textarea form element (see Figure 2-2). If you load Listing 16-10 in NN6, some of the reporting categories report "undefined" because the browser unfortunately does not pass error properties to the `handleError()` function. A Submit button is also provided to mail the bug information to a support center e-mail address — an example of how to handle the occurrence of a bug in your scripts.

Listing 16-10: **Controlling Script Errors**

```
<HTML>
<TITLE>Error Dialog Control</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
// function with invalid variable value
function goWrong() {
    var x = fred
}
// turn off error dialogs
function errOff() {
    window.onerror = doNothing
}
// turn on error dialogs with hard reload
function errOn() {
    window.onerror = handleError
}

// assign default error handler
window.onerror = handleError

// error handler when errors are turned off...prevents error dialog
function doNothing() {return true}

function handleError(msg, URL, lineNum) {
    var errWind = window.open("","errors","HEIGHT=270,WIDTH=400")
    var wintxt = "<HTML><BODY BGCOLOR=RED>"
```

```
    wintxt += "<B>An error has occurred on this page.  "
    wintxt += "Please report it to Tech Support.</B>"
    wintxt += "<FORM METHOD=POST ENCTYPE='text/plain' "
    wintxt += "ACTION=mailTo:support4@dannyg.com >"
    wintxt += "<TEXTAREA NAME='errMsg' COLS=45 ROWS=8 WRAP=VIRTUAL>"
    wintxt += "Error: " + msg + "\n"
    wintxt += "URL: " + URL + "\n"
    wintxt += "Line: " + lineNum + "\n"
    wintxt += "Client: " + navigator.userAgent + "\n"
    wintxt += "----------------------------------------\n"
    wintxt += "Please describe what you were doing when the error occurred:"
    wintxt += "</TEXTAREA><P>"
    wintxt += "<INPUT TYPE=SUBMIT VALUE='Send Error Report'>"
    wintxt += "<INPUT TYPE=button VALUE='Close' onClick='self.close()'>"
    wintxt += "</FORM></BODY></HTML>"
    errWind.document.write(wintxt)
    errWind.document.close()
    return true
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform">
<INPUT TYPE="button" VALUE="Cause an Error" onClick="goWrong()"><P>
<INPUT TYPE="button" VALUE="Turn Off Error Dialogs" onClick="errOff()">
<INPUT TYPE="button" VALUE="Turn On Error Dialogs" onClick="errOn()">
</FORM>
</BODY>
</HTML>
```

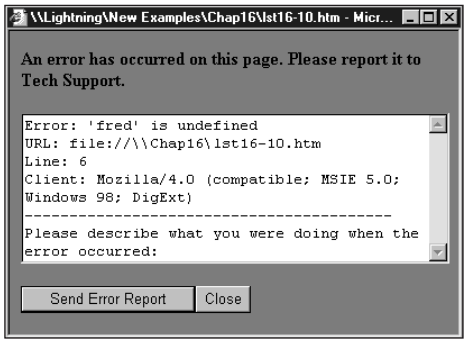


Figure 2-2: An example of a self-reporting error window

I provide a button that performs a hard reload, which, in turn, resets the `window.onerror` property to its default value. With error dialog boxes turned off, the error-handling function does not run.

opener

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓	✓	✓	✓	✓	✓

Example

To demonstrate the importance of the `opener` property, take a look at how a new window can define itself from settings in the main window (Listing 16-11). The `doNew()` function generates a small subwindow and loads the file in Listing 16-12 into the window. Notice the initial conditional statements in `doNew()` to make sure that if the new window already exists, it comes to the front by invoking the new window's `focus()` method. You can see the results in Figure 2-3. Because the `doNew()` function in Listing 16-11 uses window methods and properties not available in IE3, this example does not work correctly in IE3.

Listing 16-11: Contents of a Main Window Document That Generates a Second Window

```
<HTML>
<HEAD>
<TITLE>Master of all Windows</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
var myWind
function doNew() {
    if (!myWind || myWind.closed) {
        myWind = window.open("lst16-12.htm","subWindow",
           "HEIGHT=200,WIDTH=350,resizable")
    } else {
        // bring existing subwindow to the front
        myWind.focus()
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="input">
Select a color for a new window:
<INPUT TYPE="radio" NAME="color" VALUE="red" CHECKED>Red
<INPUT TYPE="radio" NAME="color" VALUE="yellow">Yellow
<INPUT TYPE="radio" NAME="color" VALUE="blue">Blue
<INPUT TYPE="button" NAME="storage" VALUE="Make a Window" onClick="doNew()">
<HR>
This field will be filled from an entry in another window:
<INPUT TYPE="text" NAME="entry" SIZE=25>
</FORM>
</BODY>
</HTML>
```

The `window.open()` method doesn't provide parameters for setting the new window's background color, so I let the `getColor()` function in the new window do the job as the document loads. The function uses the `opener` property to find out which radio button on the main page is selected.

Listing 16-12: References to the opener Property

```
<HTML>
<HEAD>
<TITLE>New Window on the Block</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function getColor() {
    // shorten the reference
    colorButtons = self.opener.document.forms[0].color
    // see which radio button is checked
    for (var i = 0; i < colorButtons.length; i++) {
        if (colorButtons[i].checked) {
            return colorButtons[i].value
        }
    }
    return "white"
}
</SCRIPT>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
document.write("<BODY BGCOLOR='" + getColor() + "'>")
</SCRIPT>
<H1>This is a new window.</H1>
<FORM>
<INPUT TYPE="button" VALUE="Who's in the Main window?"
onClick="alert(self.opener.document.title)"><P>
Type text here for the main window:
<INPUT TYPE="text" SIZE=25 onChange="self.opener.document.forms[0].entry.value =
this.value">
</FORM>
</BODY>
</HTML>
```

In the `getColor()` function, the multiple references to the radio button array can be very long. To simplify the references, the `getColor()` function starts out by assigning the radio button array to a variable I arbitrarily call `colorButtons`. That shorthand now stands in for lengthy references as I loop through the radio buttons to determine which button is checked and retrieve its value property.

A button in the second window simply fetches the title of the opener window's document. Even if another document loads in the main window in the meantime, the `opener` reference still points to the main window: Its `document` object, however, will change.

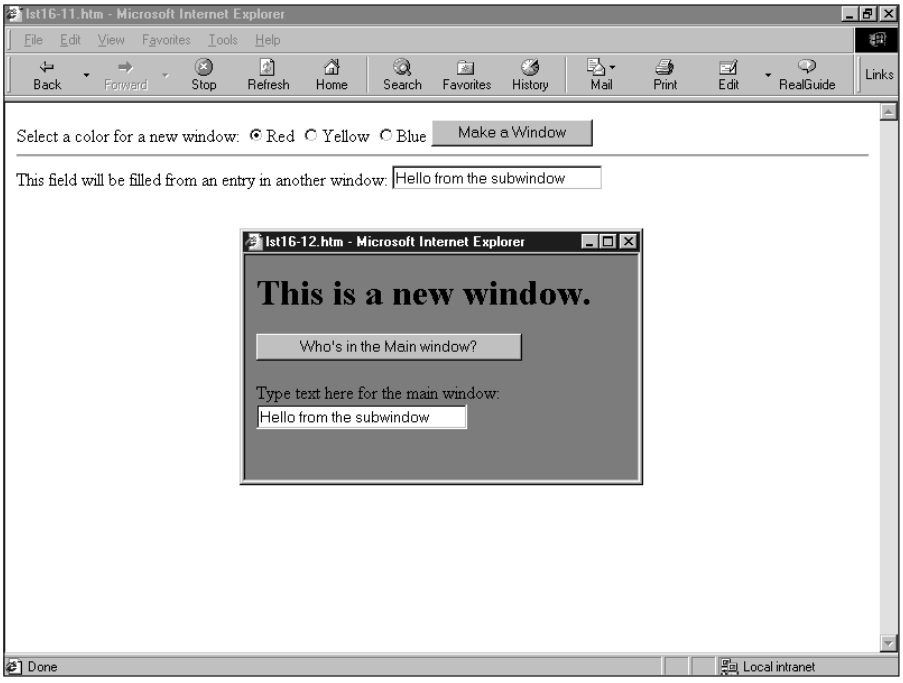


Figure 2-3: The main and subwindows, inextricably linked via the window.opener property

Finally, the second window contains a text input object. Enter any text there that you like and either tab or click out of the field. The `onChange` event handler updates the field in the opener's document (provided that document is still loaded).

pageXOffset
pageYOffset

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

The script in Listing 16-13 is an unusual construction that creates a frameset and creates the content for each of the two frames all within a single HTML document (see "Frame Object" in Chapter 16 of the *JavaScript Bible* for more details). The purpose of this example is to provide you with a playground to become familiar with the page offset concept and how the values of these properties correspond to physical activity in a scrollable document.

In the left frame of the frameset are two fields that are ready to show the pixel values of the right frame's `pageXOffset` and `pageYOffset` properties. The content

of the right frame is a 30-row table of fixed width (800 pixels). Mouse click events are captured by the document level (see Chapter 18 of the *JavaScript Bible*), allowing you to click any table or cell border or outside the table to trigger the `showOffsets()` function in the right frame. That function is a simple script that displays the page offset values in their respective fields in the left frame.

Listing 16-13: Viewing the pageXOffset and pageYOffset Properties

```
<HTML>
<HEAD>
<TITLE>Master of all Windows</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function leftFrame() {
    var output = "<HTML><BODY><H3>Page Offset Values</H3><HR>\n"
    output += "<FORM>PageXOffset:<INPUT TYPE='text' NAME='xOffset' SIZE=4><BR>\n"
    output += "PageYOffset:<INPUT TYPE='text' NAME='yOffset' SIZE=4><BR>\n"
    output += "</FORM></BODY></HTML>"
    return output
}

function rightFrame() {
    var output = "<HTML><HEAD><SCRIPT LANGUAGE='JavaScript'>\n"
    output += "function showOffsets() {\n"
    output += "parent.readout.document.forms[0].xOffset.value =
self.pageXOffset\n"
    output += "parent.readout.document.forms[0].yOffset.value =
self.pageYOffset\n}\n"
    output += "document.captureEvents(Event.CLICK)\n"
    output += "document.onclick = showOffsets\n"
    output += "<\/SCRIPT></HEAD><BODY><H3>Content Page</H3>\n"
    output += "Scroll this frame and click on a table border to view " +
              "page offset values.<BR><HR>\n"
    output += "<TABLE BORDER=5 WIDTH=800>"
    var oneRow = "<TD>Cell 1</TD><TD>Cell 2</TD><TD>Cell 3</TD>" +
                 "<TD>Cell 4</TD><TD>Cell 5</TD>"
    for (var i = 1; i <= 30; i++) {
        output += "<TR><TD><B>Row " + i + "</B></TD>" + oneRow + "</TR>"
    }
    output += "</TABLE></BODY></HTML>"
    return output
}
</SCRIPT>
</HEAD>
<FRAMESET COLS="30%,70%">
    <FRAME NAME="readout" SRC="javascript:parent.leftFrame()">
    <FRAME NAME="display" SRC="javascript:parent.rightFrame()">
</FRAMESET>
</HTML>
```

To gain an understanding of how the offset values work, scroll the window slightly in the horizontal direction and notice that the `pageXOffset` value increases; the same goes for the `pageYOffset` value as you scroll down. Remember that these values reflect the coordinate in the document that is currently under the top-left corner of the window (frame) holding the document. You can see an IE4+ version of this example in Listing 18-20 (in Chapter 4 of this book). A cross-browser version would require very little browser branching.

parent

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

To demonstrate how various `window` object properties refer to window levels in a multiframe environment, use your browser to load the Listing 16-14 document. It, in turn, sets each of two equal-size frames to the same document: Listing 16-15. This document extracts the values of several window properties, plus the `document.title` properties of two different window references.

Listing 16-14: **Framesetting Document for Listing 16-15**

```
<HTML>
<HEAD>
<TITLE>The Parent Property Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
self.name = "Framesetter"
</SCRIPT>
</HEAD>
<FRAMESET COLS="50%,50%" onUnload="self.name = ''">
    <FRAME NAME="JustAKid1" SRC="lst16-15.htm">
    <FRAME NAME="JustAKid2" SRC="lst16-15.htm">
</FRAMESET>
</HTML>
```

Listing 16-15: **Revealing Various Window-Related Properties**

```
<HTML>
<HEAD>
<TITLE>Window Revealer II</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function gatherWindowData() {
    var msg = ""
```

```
    msg = msg + "top name: " + top.name + "<BR>"
    msg = msg + "parent name: " + parent.name + "<BR>"
    msg = msg + "parent.document.title: " + parent.document.title + "<P>"
    msg = msg + "window name: " + window.name + "<BR>"
    msg = msg + "self name: " + self.name + "<BR>"
    msg = msg + "self.document.title: " + self.document.title
    return msg
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write(gatherWindowData())
</SCRIPT>
</BODY>
</HTML>
```

In the two frames (Figure 2-4), the references to the `window` and `self` object names return the name assigned to the frame by the frameset definition (`JustAKid1` for the left frame, `JustAKid2` for the right frame). In other words, from each frame's point of view, the `window` object is its own frame. References to `self.document.title` refer only to the document loaded into that window frame. But references to the top and parent windows (which are one and the same in this example) show that those object properties are shared between both frames.

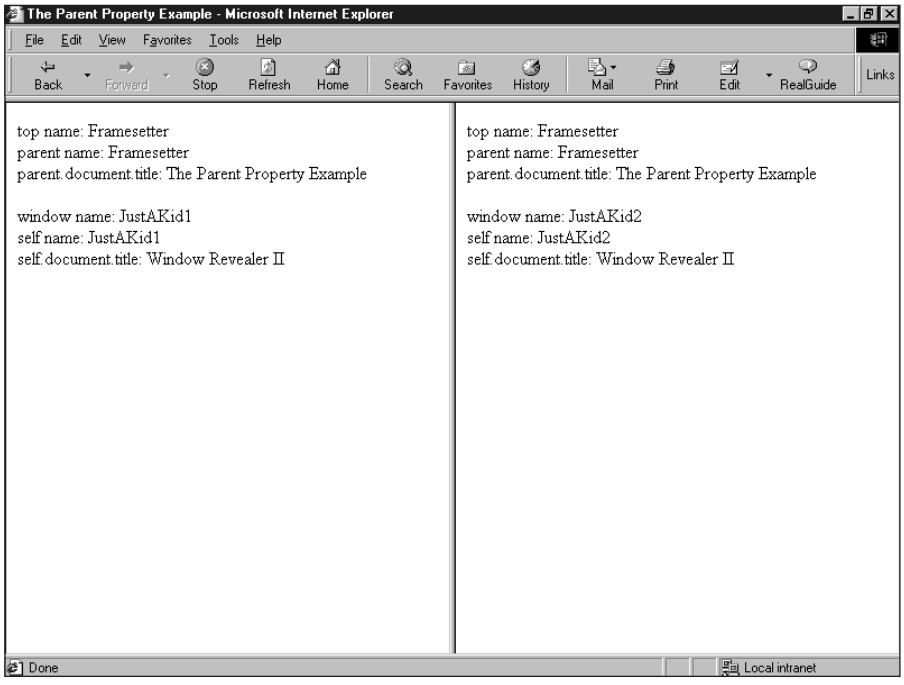


Figure 2-4: Parent and top properties being shared by both frames

A couple other fine points are worth highlighting. First, the name of the framesetting window is set as Listing 16-14 loads, rather than in response to an `onLoad` event handler in the `<FRAMESET>` tag. The reason for this is that the name must be set in time for the documents loading in the frames to get that value. If I had waited until the frameset's `onLoad` event handler, the name wouldn't be set until after the frame documents had loaded. Second, I restore the parent window's name to an empty string when the framesetting document unloads. This is to prevent future pages from getting confused about the window name.

returnValue

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 16-39 for the `showModalDialog()` method for an example of how to get data back from a dialog box in IE4+.

screenLeft
screenTop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `screenLeft` and `screenTop` properties. Start with the browser window maximized (if you are using Windows). Enter the following property name into the top text box:

```
window.screenLeft
```

Click the Evaluate button to see the current setting. Unmaximize the window and drag it around the screen. Each time you finish dragging, click the Evaluate button again to see the current value. Do the same for `window.screenTop`.

screenX
screenY

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `screenX` and `screenY` properties in NN6. Start with the browser window maximized (if you are using Windows). Enter the following property name into the top text box:

```
window.screenY
```

Click the Evaluate button to see the current setting. Unmaximize the window and drag it around the screen. Each time you finish dragging, click the Evaluate button again to see the current value. Do the same for `window.screenY`.

scrollX
scrollY

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `scrollX` and `scrollY` properties in NN6. Enter the following property into the top text box:

```
window.scrollY
```

Now manually scroll the page down so that you can still see the Evaluate button. Click the button to see how far the window has scrolled along the y-axis.

self

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 16-16 uses the same operations as Listing 16-5 but substitutes the `self` property for all `window` object references. The application of this reference is entirely optional, but it can be helpful for reading and debugging scripts if the HTML document is to appear in one frame of a multiframe window — especially if other JavaScript code in this document refers to documents in other frames. The `self` reference helps anyone reading the code know precisely which frame was being addressed.

Listing 16-16: **Using the self Property**

```
<HTML>
<HEAD>
<TITLE>self Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
self.defaultStatus = "Welcome to my Web site."
</SCRIPT>
</HEAD>
<BODY>
<A HREF="http:// www.microsoft.com"
onMouseOver="self.status = 'Visit Microsoft\'s Home page.';return true"
onMouseOut="self.status = '';return true">Microsoft</A><P>
<A HREF="http://home.netscape.com"
onMouseOver="self.status = 'Visit Netscape\'s Home page.';return true"
onMouseOut="self.status = self.defaultStatus;return true">Netscape</A>
</BODY>
</HTML>
```

status

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

In Listing 16-17, the `status` property is set in a handler embedded in the `onMouseOver` attribute of two HTML link tags. Notice that the handler requires a `return true` statement (or any expression that evaluates to `return true`) as the last statement of the handler. This statement is required or the status message will not display, particularly in early browsers.

Listing 16-17: **Links with Custom Statusbar Messages**

```
<HTML>
<HEAD>
<TITLE>window.status Property</TITLE>
</HEAD>
<BODY>
<A HREF="http://www.dannyg.com" onMouseOver="window.status = 'Go to my Home
page. (www.dannyg.com)'; return true">Home</A><P>
<A HREF="http://home.netscape.com" onMouseOver="window.status = 'Visit Netscape
Home page. (home.netscape.com)'; return true">Netscape</A>
</BODY>
</HTML>
```

As a safeguard against platform-specific anomalies that affect the behavior of `onMouseOver` event handlers and the `window.status` property, you should also include an `onMouseOut` event handler for links and client-side image map area objects. Such `onMouseOut` event handlers should set the `status` property to an empty string. This setting ensures that the statusbar message returns to the `defaultStatus` setting when the pointer rolls away from these objects. If you want to write a generalizable function that handles all window status changes, you can do so, but word the `onMouseOver` attribute carefully so that the event handler evaluates to `return true`. Listing 16-18 shows such an alternative.

Listing 16-18: **Handling Status Message Changes**

```
<HTML>
<HEAD>
<TITLE>Generalizable window.status Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function showStatus(msg) {
    window.status = msg
    return true
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="http:// www.dannyg.com " onMouseOver="return showStatus('Go to my Home
page (www.dannyg.com).')" onMouseOut="return showStatus('')">Home</A><P>
<A HREF="http://home.netscape.com" onMouseOver="return showStatus('Visit
Netscape Home page.')" onMouseOut="return showStatus('')">Netscape</A>
</BODY>
</HTML>
```

Notice how the event handlers return the results of the `showStatus()` method to the event handler, allowing the entire handler to evaluate to `return true`.

One final example of setting the statusbar (shown in Listing 16-19) also demonstrates how to create a simple scrolling banner in the statusbar.

Listing 16-19: **Creating a Scrolling Banner**

```
<HTML>
<HEAD>
<TITLE>Message Scroller</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var msg = "Welcome to my world..."
var delay = 150
var timerId
var maxCount = 0
var currCount = 1
```

Continued

Listing 16-19 *(continued)*

```
function scrollMsg() {
    // set the number of times scrolling message is to run
    if (maxCount == 0) {
        maxCount = 3 * msg.length
    }
    window.status = msg
    // keep track of how many characters have scrolled
    currCount++
    // shift first character of msg to end of msg
    msg = msg.substring (1, msg.length) + msg.substring (0, 1)
    // test whether we've reached maximum character count
    if (currCount >= maxCount) {
        timerID = 0        // zero out the timer
        window.status = ""    // clear the status bar
        return            // break out of function
    } else {
        // recursive call to this function
        timerId = setTimeout("scrollMsg()", delay)
   }
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="scrollMsg()">
</BODY>
</HTML>
```

Because the statusbar is being set by a standalone function (rather than by an `onMouseOver` event handler), you do not have to append a `return true` statement to set the `status` property. The `scrollMsg()` function uses more advanced JavaScript concepts, such as the `window.setTimeout()` method (covered later in this chapter) and string methods (covered in Chapter 34 of the *JavaScript Bible*). To speed the pace at which the words scroll across the statusbar, reduce the value of `delay`.

Many Web surfers (myself included) don't care for these scrollers that run forever in the statusbar. Rolling the mouse over links disturbs the banner display. Scrollers can also crash earlier browsers, because the `setTimeout()` method eats application memory in Navigator 2. Use scrolling bars sparingly or design them to run only a few times after the document loads.

Setting the `status` property with `onMouseOver` event handlers has had a checkered career along various implementations in Navigator. A script that sets the statusbar is always in competition against the browser itself, which uses the statusbar to report loading progress. When a "hot" area on a page is at the edge of a frame, many times the `onMouseOut` event fails to fire, thus preventing the statusbar from clearing itself. Be sure to torture test any such implementations before declaring your page ready for public access.

Methods

`alert("message")`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The parameter for the example in Listing 16-20 is a concatenated string. It joins together two fixed strings and the value of the browser's `navigator.appName` property. Loading this document causes the alert dialog box to appear, as shown in several configurations in Figure 2-5. The JavaScript Alert: line cannot be deleted from the dialog box in earlier browsers, nor can the title bar be changed in later browsers.

Listing 16-20: Displaying an Alert Dialog Box

```
<HTML>
<HEAD>
<TITLE>window.alert() Method</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
alert("You are running the " + navigator.appName + " browser.")
</SCRIPT>
</BODY>
</HTML>
```

Figure 2-5: Results of the alert() method in Listing 16-20 in Internet Explorer 5 (top) and Navigator 6 (bottom) for Windows 98

captureEvents(*eventTypeList*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

The page in Listing 16-21 is an exercise in capturing and releasing click events in the `window` object. Whenever the window is capturing click events, the `flash()` function runs. In that function, the event is examined so that only if the Control key is also being held down and the name of the button starts with "button" does the document background color flash red. For all click events (that is, those directed at objects on the page capable of their own `onClick` event handlers), the click is processed with the `routeEvent()` method to make sure the target buttons execute their own `onClick` event handlers.

Listing 16-21: Capturing Click Events in the Window

```
<HTML>
<HEAD>
<TITLE>Window Event Capture</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
// function to run when window captures a click event
function flash(e) {
    if (e.modifiers = Event.CONTROL_MASK &&
    e.target.name.indexOf("button") == 0) {
        document.bgColor = "red"
        setTimeout("document.bgColor = 'white'", 500)
    }
    // let event continue to target
    routeEvent(e)
}
// default setting to capture click events
window.captureEvents(Event.CLICK)
// assign flash() function to click events captured by window
window.onclick = flash
</SCRIPT>
</HEAD>
<BODY BGCOLOR="white">
<FORM NAME="buttons">
<B>Turn window click event capture on or off (Default is "On")</B><P>
<INPUT NAME="captureOn" TYPE="button" VALUE="Capture On"
onClick="window.captureEvents(Event.CLICK)"> 
<INPUT NAME="captureOff" TYPE="button" VALUE="Capture Off"
onClick="window.releaseEvents(Event.CLICK)">
<HR>
<B>Ctrl+Click on a button to see if clicks are being captured by the window
(background color will flash red):</B><P>
<UL>
```

```
<LI><INPUT NAME="button1" TYPE="button" VALUE="Informix" onClick="alert('You
clicked on Informix.')">
<LI><INPUT NAME="button2" TYPE="button" VALUE="Oracle" onClick="alert('You
clicked on Oracle.')">
<LI><INPUT NAME="button3" TYPE="button" VALUE="Sybase" onClick="alert('You
clicked on Sybase.')">
</UL>
</FORM>
</BODY>
</HTML>
```

When you try this page, also turn off window event capture. Now only the buttons' `onClick` event handlers execute, and the page does not flash red.

clearInterval(*intervalIDnumber*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

See Listings 16-36 and 16-37 for an example of how `setInterval()` and `clearInterval()` are used together on a page.

clearTimeout(*timeoutIDnumber*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The page in Listing 16-22 features one text field and two buttons (Figure 2-6). One button starts a countdown timer coded to last one minute (easily modifiable for other durations); the other button interrupts the timer at any time while it is running. When the minute is up, an alert dialog box lets you know.

Listing 16-22: **A Countdown Timer**

```
<HTML>
<HEAD>
<TITLE>Count Down Timer</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var running = false
```

Continued

Listing 16-22 *(continued)*

```
var endTime = null
var timerID = null

function startTimer() {
    running = true
    now = new Date()
    now = now.getTime()
    // change last multiple for the number of minutes
    endTime = now + (1000 * 60 * 1)
    showCountDown()
}

function showCountDown() {
    var now = new Date()
    now = now.getTime()
    if (endTime - now <= 0) {
        stopTimer()
        alert("Time is up.  Put down your pencils.")
    } else {
        var delta = new Date(endTime - now)
        var theMin = delta.getMinutes()
        var theSec = delta.getSeconds()
        var theTime = theMin
        theTime += ((theSec < 10) ? ":0" : ":") + theSec
        document.forms[0].timerDisplay.value = theTime
        if (running) {
            timerID = setTimeout("showCountDown()",1000)
        }
    }
}

function stopTimer() {
    clearTimeout(timerID)
    running = false
    document.forms[0].timerDisplay.value = "0:00"
}
//-->
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button" NAME="startTime" VALUE="Start 1 min. Timer"
onClick="startTimer()">
<INPUT TYPE="button" NAME="clearTime" VALUE="Clear Timer"
onClick="stopTimer()"><P>
<INPUT TYPE="text" NAME="timerDisplay" VALUE="">
</FORM>
</BODY>
</HTML>
```

Notice that the script establishes three variables with global scope in the window: `running`, `endTime`, and `timerID`. These values are needed inside multiple functions, so they are initialized outside of the functions.

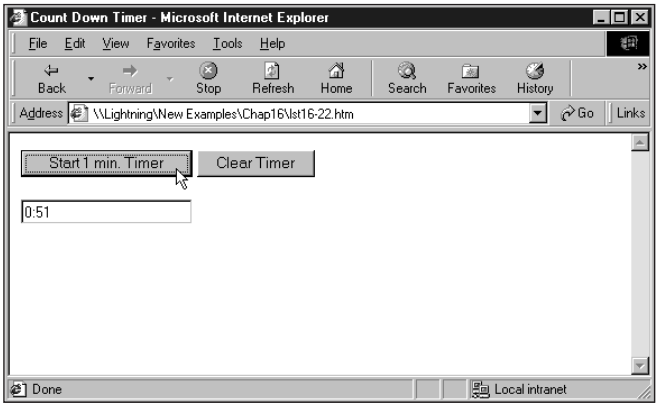


Figure 2-6: The countdown timer page as it displays the time remaining

In the `startTimer()` function, you switch the running flag on, meaning that the timer should be going. Using some date functions (see Chapter 36 of the *JavaScript Bible*), you extract the current time in milliseconds and add the number of milliseconds for the next minute (the extra multiplication by one is the place where you can change the amount to the desired number of minutes). With the end time stored in a global variable, the function now calls another function that compares the current and end times and displays the difference in the text field.

Early in the `showCountDown()` function, check to see if the timer has wound down. If so, you stop the timer and alert the user. Otherwise, the function continues to calculate the difference between the two times and formats the time in mm:ss format. As long as the `running` flag is set to `true`, the function sets the one-second timeout timer before repeating itself. To stop the timer before it has run out (in the `stopTimer()` function), the most important step is to cancel the timeout running inside the browser. The `clearTimeout()` method uses the global `timerID` value to do that. Then the function turns off the running switch and zeros out the display.

When you run the timer, you may occasionally notice that the time skips a second. It's not cheating. It just takes slightly more than one second to wait for the timeout and then finish the calculations for the next second's display. What you're seeing is the display catching up with the real time left.

close()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listing 16-4 (for the `window.closed` property), which provides an elaborate, cross-platform, bug-accommodating example of applying the `window.close()` method across multiple windows.

`confirm("message")`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The example in Listing 16-23 shows the user interface part of how you can use a confirm dialog box to query a user before clearing a table full of user-entered data. The line in the title bar, as shown in Figure 2-7, or the "JavaScript Confirm" legend in earlier browser versions, cannot be removed from the dialog box.

Listing 16-23: **The Confirm Dialog Box**

```
<HTML>
<HEAD>
<TITLE>window.confirm() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function clearTable() {
    if (confirm("Are you sure you want to empty the table?")) {
        alert("Emptying the table...") // for demo purposes
        //statements that actually empty the fields
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<!-- other statements that display and populate a large table -->
<INPUT TYPE="button" NAME="clear" VALUE="Reset Table" onClick="clearTable()">
</FORM>
</BODY>
</HTML>
```

Figure 2-7: A JavaScript confirm dialog box (IE5/Windows format)

createPopup()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

See Listing 16-49 later in this chapter for an example of the `createPopup()` method.

disableExternalCapture()
enableExternalCapture()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

As this was a little-used feature of NN4 even while the browser enjoyed a substantial installed base, it becomes less important as that browser version recedes into history. You can find an example of this feature at the Support Center for this book (`http://www.dannyg.com/update.html`) or on pp.213–214 of the *JavaScript Bible,* 3rd edition.

execScript("*exprList*"[, *language*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `execScript()` method. The Evaluator has predeclared global variables for the lowercase letters `a` through `z`. Enter each of the following statements into the top text box and observe the results for each.

```
a
```

When first loaded, the variable is declared but assigned no value, so it is undefined.

```
window.execScript("a = 5")
```

The method returns no value, so the mechanism inside The Evaluator says that the statement is undefined.

```
a
```

The variable is now 5.

```
window.execScript("b = a * 50")
b
```

The `b` global variable has a value of 250. Continue exploring with additional script statements. Use semicolons to separate multiple statements within the string parameter.

find(["*searchString*" [, *matchCaseBoolean*, *searchUpBoolean*]])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

A simple call to the `window.find()` method looks as follows:

```
var success = window.find("contract")
```

If you want the search to be case-sensitive, add at least one of the two optional parameters:

```
success = wind.find(matchString,caseSensitive,backward)
```

Because this method works only in NN4, refer to discussions of the `TextRange` and `Range` objects in Chapter 19 of the *JavaScript Bible* for more modern implementations of body text searching.

GetAttention()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) in NN6 to set a timer that gives you enough time to switch to another application and wait for the attention signal to fire. Enter the following statement into the top text box, click the Evaluate button, and then quickly switch to another program:

```
setTimeout("GetAttention()", 5000)
```

After a total of five seconds, the attention signal fires.

moveBy(*deltaX*,*deltaY*)
moveTo(*x*,*y*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Several examples of using the `window.moveTo()` and `window.moveBy()` methods are shown in Listing 16-24. The page presents four buttons, each of which performs a different kind of browser window movement.

Listing 16-24: **Window Boogie**

```
<HTML>
<HEAD>
<TITLE>Window Gymnastics</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
var isNav4 = ((navigator.appName == "Netscape") &&
(parseInt(navigator.appVersion) >= 4))
// wait in onLoad for page to load and settle in IE
function init() {
    // fill missing IE properties
    if (!window.outerWidth) {
        window.outerWidth = document.body.clientWidth
        window.outerHeight = document.body.clientHeight + 30
    }
    // fill missing IE4 properties
    if (!screen.availWidth) {
        screen.availWidth = 640
        screen.availHeight = 480
    }
}
// function to run when window captures a click event
function moveOffScreen() {
    // branch for NN security
    if (isNav4) {
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite")
    }
    var maxX = screen.width
    var maxY = screen.height
    window.moveTo(maxX+1, maxY+1)
    setTimeout("window.moveTo(0,0)",500)
    if (isNav4) {
netscape.security.PrivilegeManager.disablePrivilege("UniversalBrowserWrite")
    }
```

Continued

Listing 16-24 *(continued)*

```
}
// moves window in a circular motion
function revolve() {
    var winX = (screen.availWidth - window.outerWidth) / 2
    var winY = 50
    window.resizeTo(400,300)
    window.moveTo(winX, winY)

    for (var i = 1; i < 36; i++) {
        winX += Math.cos(i * (Math.PI/18)) * 5
        winY += Math.sin(i * (Math.PI/18)) * 5
        window.moveTo(winX, winY)
    }
}
// moves window in a horizontal zig-zag pattern
function zigzag() {
    window.resizeTo(400,300)
    window.moveTo(0,80)
    var incrementX = 2
    var incrementY = 2
    var floor = screen.availHeight - window.outerHeight
    var rightEdge = screen.availWidth - window.outerWidth
    for (var i = 0; i < rightEdge; i += 2) {
        window.moveBy(incrementX, incrementY)
        if (i%60 == 0) {
            incrementY = -incrementY
        }
    }
}
// resizes window to occupy all available screen real estate
function maximize() {
    window.moveTo(0,0)
    window.resizeTo(screen.availWidth, screen.availHeight)
}
</SCRIPT>
</HEAD>
<BODY onLoad="init()">
<FORM NAME="buttons">
<B>Window Gymnastics</B><P>
<UL>
<LI><INPUT NAME="offscreen" TYPE="button" VALUE="Disappear a Second"
onClick="moveOffScreen()">
<LI><INPUT NAME="circles" TYPE="button" VALUE="Circular Motion"
onClick="revolve()">
<LI><INPUT NAME="bouncer" TYPE="button" VALUE="Zig Zag" onClick="zigzag()">
<LI><INPUT NAME="expander" TYPE="button" VALUE="Maximize" onClick="maximize()">
</UL>
</FORM>
</BODY>
</HTML>
```

To run successfully in NN, the first button requires that you have codebase principals turned on (see Chapter 46 of the *JavaScript Bible*) to take advantage of what would normally be a signed script. The `moveOffScreen()` function momentarily moves the window entirely out of view. Notice how the script determines the size of the screen before deciding where to move the window. After the journey off screen, the window comes back into view at the upper-left corner of the screen.

If using the Web sometimes seems like going around in circles, then the second function, `revolve()`, should feel just right. After reducing the size of the window and positioning it near the top center of the screen, the script uses a bit of math to position the window along 36 places around a perfect circle (at 10-degree increments). This is an example of how to control a window's position dynamically based on math calculations. IE complicates the job a bit by not providing properties that reveal the outside dimensions of the browser window.

To demonstrate the `moveBy()` method, the third function, `zigzag()`, uses a `for` loop to increment the coordinate points to make the window travel in a saw tooth pattern across the screen. The x coordinate continues to increment linearly until the window is at the edge of the screen (also calculated on the fly to accommodate any size monitor). The y coordinate must increase and decrease as that parameter changes direction at various times across the screen.

In the fourth function, you see some practical code (finally) that demonstrates how best to simulate maximizing the browser window to fill the entire available screen space on the visitor's monitor.

navigate("*URL*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility					✓	✓	✓	✓	✓

Example

Supply any valid URL as the parameter to the method, as in

```
window.navigate("http://www.dannyg.com")
```

open("*URL*", "*windowName*" [, "*windowFeatures*"][,*replaceFlag*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The page rendered by Listing 16-26 displays a single button that generates a new window of a specific size that has only the statusbar turned on. The script here

shows all the elements necessary to create a new window that has all the right stuff on most platforms. The new window object reference is assigned to a global variable, `newWindow`. Before a new window is generated, the script looks to see if the window has never been generated before (in which case `newWindow` would be `null`) or, for newer browsers, the window is closed. If either condition is true, the window is created with the `open()` method. Otherwise, the existing window is brought forward with the `focus()` method (NN3+ and IE4+).

As a safeguard against older browsers, the script manually adds an `opener` property to the new window if one is not already assigned by the `open()` method. The current `window` object reference is assigned to that property.

Due to the timing problem that afflicts all IE generations, the HTML assembly and writing to the new window is separated into its own function that is invoked after a 50 millisecond delay (NN goes along for the ride, but it could accommodate the assembly and writing without the delay). To build the string that is eventually written to the document, I use the += (add-by-value) operator, which appends the string on the right side of the operator to the string stored in the variable on the left side. In this example, the new window is handed an `<H1>`-level line of text to display.

Listing 16-26: **Creating a New Window**

```
<HTML>
<HEAD>
<TITLE>New Window</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var newWindow
function makeNewWindow() {
    if (!newWindow || newWindow.closed) {
        newWindow = window.open("","","status,height=200,width=300")
        if (!newWindow.opener) {
        newWindow.opener = window
        }
        // force small delay for IE to catch up
        setTimeout("writeToWindow()", 50)
    } else {
        // window's already open; bring to front
        newWindow.focus()
    }
}
function writeToWindow() {
    // assemble content for new window
    var newContent = "<HTML><HEAD><TITLE>One Sub Window</TITLE></HEAD>"
    newContent += "<BODY><H1>This window is brand new.</H1>"
    newContent += "</BODY></HTML>"
    // write HTML to new window document
    newWindow.document.write(newContent)
    newWindow.document.close() // close layout stream
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
```

```
<INPUT TYPE="button" NAME="newOne" VALUE="Create New Window"
 onClick="makeNewWindow()">
</FORM>
</BODY>
</HTML>
```

If you need to create a new window for the lowest common denominator of scriptable browser, you will have to omit the `focus()` method and the `window.closed` property from the script (as well as add the NN2 bug workaround described earlier). Or you may prefer to forego a subwindow for all browsers below a certain level. See Listing 16-3 (in the `window.closed` property discussion) for other ideas about cross-browser authoring for subwindows.

print()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓				✓	✓

Example

Listing 16-27 is a frameset that loads Listing 16-28 into the top frame and a copy of the Bill of Rights into the bottom frame.

Listing 16-27: **Print Frameset**

```
<HTML>
<HEAD>
<TITLE>window.print() method</TITLE>
</HEAD>
<FRAMESET ROWS="25%,75%">
    <FRAME NAME="controls" SRC="lst16-28.htm">
    <FRAME NAME="display" SRC="bofright.htm">
</FRAMESET>
</HTML>
```

Two buttons in the top control panel (Listing 16-28) let you print the whole frameset (in those browsers and OSs that support it) or just the lower frame. To print the entire frameset, the reference includes the parent window; to print the lower frame, the reference is directed at the `parent.display` frame.

Listing 16-28: **Printing Control**

```
<HTML>
<HEAD>
<TITLE>Print()</TITLE>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" NAME="printWhole" VALUE="Print Entire Frameset"
onClick="parent.print()"><P>
<INPUT TYPE="button" NAME="printFrame" VALUE="Print Bottom Frame Only"
onClick="parent.display.print()"><P>
</FORM>
</BODY>
</HTML>
```

If you don't like some facet of the printed output, blame the browser's print engine, and not JavaScript. The `print()` method merely invokes the browser's regular printing routines. Pages whose content is generated entirely by JavaScript print only in NN3+ and IE4+.

`prompt("message", "defaultReply")`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The function that receives values from the prompt dialog box in Listing 16-29 (see the dialog box in Figure 2-8) does some data-entry validation (but certainly not enough for a commercial site). The function first checks to make sure that the returned value is neither `null` (Cancel) nor an empty string (the user clicked OK without entering any values). See Chapter 43 of the *JavaScript Bible* for more about data-entry validation.

Listing 16-29: **The Prompt Dialog Box**

```
<HTML>
<HEAD>
<TITLE>window.prompt() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function populateTable() {
    var howMany = prompt("Fill in table for how many factors?","")
    if (howMany != null && howMany != "") {
```

```
        alert("Filling the table for " + howMany) // for demo
        //statements that validate the entry and
        //actually populate the fields of the table
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<!-- other statements that display and populate a large table -->
<INPUT TYPE="button" NAME="fill" VALUE="Fill Table..."
onClick="populateTable()">
</FORM>
</BODY>
</HTML>
```

Figure 2-8: The prompt dialog box displayed from Listing 16-29 (Windows format)

Notice one important user interface element in Listing 16-29. Because clicking the button leads to a dialog box that requires more information from the user, the button's label ends in an ellipsis (or, rather, three periods acting as an ellipsis character). The ellipsis is a common courtesy to let users know that a user interface element leads to a dialog box of some sort. As in similar situations in Windows and Macintosh programs, the user should be able to cancel out of that dialog box and return to the same screen state that existed before the button was clicked.

resizeBy(*deltaX*,*deltaY*)
resizeTo(*outerwidth*,*outerheight*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

You can experiment with the resize methods with the page in Listing 16-30. Two parts of a form let you enter values for each method. The one for `window.resize()` also lets you enter a number of repetitions to better see the impact of the values. Enter zero and negative values to see how those affect the method. Also test the limits of different browsers.

Listing 16-30: **Window Resize Methods**

```
<HTML>
<HEAD>
<TITLE>Window Resize Methods</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doResizeBy(form) {
    var x = parseInt(form.resizeByX.value)
    var y = parseInt(form.resizeByY.value)
    var count = parseInt(form.count.value)
    for (var i = 0; i < count; i++) {
        window.resizeBy(x, y)
    }
}
function doResizeTo(form) {
    var x = parseInt(form.resizeToX.value)
    var y = parseInt(form.resizeToY.value)
    window.resizeTo(x, y)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<B>Enter the x and y increment, plus how many times the window should be resized
by these increments:</B><BR>
Horiz:<INPUT TYPE="text" NAME="resizeByX" SIZE=4>
Vert:<INPUT TYPE="text" NAME="resizeByY" SIZE=4>
How Many:<INPUT TYPE="text" NAME="count" SIZE=4>
<INPUT TYPE="button" NAME="ResizeBy" VALUE="Show resizeBy()"
onClick="doResizeBy(this.form)">
<HR>
<B>Enter the desired width and height of the current window:</B><BR>
Width:<INPUT TYPE="text" NAME="resizeToX" SIZE=4>
Height:<INPUT TYPE="text" NAME="resizeToY" SIZE=4>
<INPUT TYPE="button" NAME="ResizeTo" VALUE="Show resizeTo()"
onClick="doResizeTo(this.form)">
</FORM>
</BODY>
</HTML>
```

routeEvent(*event*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

The `window.routeEvent()` method is used in the example for `window.captureEvents()`, Listing 16-21.

scroll(*horizontalCoord, verticalCoord*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

To demonstrate the `scroll()` method, Listing 16-31 defines a frameset with a document in the top frame (Listing 16-32) and a control panel in the bottom frame (Listing 16-33). A series of buttons and text fields in the control panel frame directs the scrolling of the document. I've selected an arbitrary, large GIF image to use in the example. To see results of some horizontal scrolling values, you may need to shrink the width of the browser window until a horizontal scrollbar appears in the top frame. Figure 2-9 shows the results in a shrunken window with modest horizontal and vertical scroll values entered into the bottom text boxes. If you substitute `scrollTo()` for the `scroll()` methods in Listing 16-33, the results will be the same, but you will need version browsers at a minimum to run it.

Listing 16-31: **A Frameset for the scroll() Demonstration**

```
<HTML>
<HEAD>
<TITLE>window.scroll() Method</TITLE>
</HEAD>

<FRAMESET ROWS="50%,50%">
    <FRAME SRC="lst16-32.htm" NAME="display">
    <FRAME SRC="lst16-33.htm" NAME="control">
</FRAMESET>
</HTML>
```

Listing 16-32: **The Image to Be Scrolled**

```
<HTML>
<HEAD>
<TITLE>Arch</TITLE>
</HEAD>
```

Continued

Listing 16-32 *(continued)*

```
<BODY>
<H1>A Picture is Worth...</H1>
<HR>
<CENTER>
<TABLE BORDER=3>
<CAPTION ALIGN=bottom>A Splendid Arch</CAPTION>
<TD>
<IMG SRC="arch.gif">
</TD></TABLE></CENTER>
</BODY>
</HTML>
```

Listing 16-33: **Controls to Adjust Scrolling of the Upper Frame**

```
<HTML>
<HEAD>
<TITLE>Scroll Controller</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
function scroll(x,y) {
    parent.frames[0].scroll(x,y)
}
function customScroll(form) {
    parent.frames[0].scroll(parseInt(form.x.value),parseInt(form.y.value))
}
</SCRIPT>
</HEAD>
<BODY>
<H2>Scroll Controller</H2>
<HR>
<FORM NAME="fixed">
Click on a scroll coordinate for the upper frame:<P>
<INPUT TYPE="button" VALUE="0,0" onClick="scroll(0,0)">
<INPUT TYPE="button" VALUE="0,100" onClick="scroll(0,100)">
<INPUT TYPE="button" VALUE="100,0" onClick="scroll(100,0)">
<P>
<INPUT TYPE="button" VALUE="-100,100" onClick="scroll(-100,100)">
<INPUT TYPE="button" VALUE="20,200" onClick="scroll(20,200)">
<INPUT TYPE="button" VALUE="1000,3000" onClick="scroll(1000,3000)">
</FORM>
<HR>
<FORM NAME="custom">
Enter a Horizontal
<INPUT TYPE="text" NAME="x" VALUE="0" SIZE=4>
and Vertical
<INPUT TYPE="text" NAME="y" VALUE="0" SIZE=4>
value.  Then
```

```
<INPUT TYPE="button" VALUE="click to scroll" onClick="customScroll(this.form)">
</FORM>
</BODY>
</HTML>
```

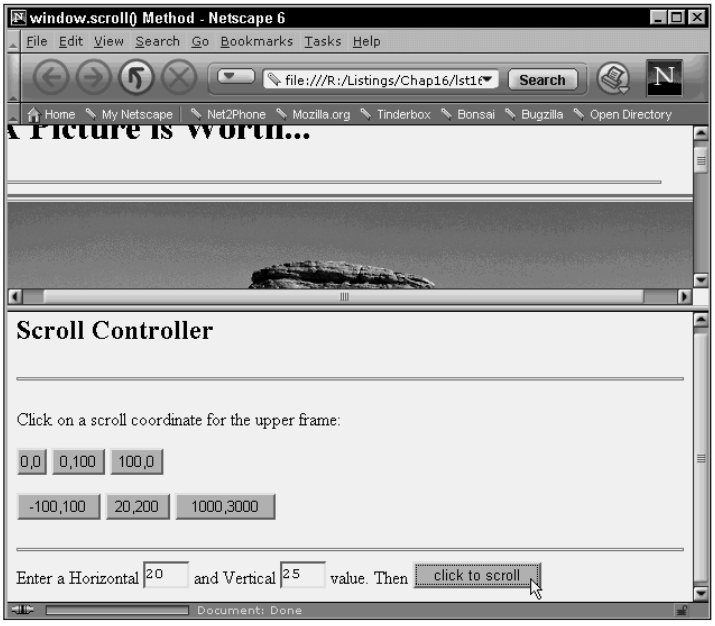


Figure 2-9: Scripts control the scrolling of the top frame

Notice that in the `customScroll()` function, JavaScript must convert the string values from the two text boxes to integers (with the `parseInt()` method) for the `scroll()` method to accept them. Nonnumeric data can produce very odd results. Also be aware that although this example shows how to adjust the scroll values in another frame, you can set such values in the same frame or window as the script, as well as in subwindows, provided that you use the correct object references to the window.

scrollBy(*deltaX*,*deltaY*)
scrollTo(*x*,*y*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

To work with the `scrollTo()` method, you can use Listings 16-31 through 16-33 (the `window.scroll()` method) but substitute `window.scrollTo()` for `window.scroll()`. The results should be the same. For `scrollBy()`, the example starts with the frameset in Listing 16-34. It loads the same content document as the `window.scroll()` example (Listing 16-32), but the control panel (Listing 16-35) provides input to experiment with the `scrollBy()` method.

Listing 16-34: **Frameset for ScrollBy Controller**

```
<HTML>
<HEAD>
<TITLE>window.scrollBy() Method</TITLE>
</HEAD>

<FRAMESET ROWS="50%,50%">
    <FRAME SRC="lst16-32.htm" NAME="display">
    <FRAME SRC="lst16-35.htm" NAME="control">
</FRAMESET>
</HTML>
```

Notice in Listing 16-35 that all references to window properties and methods are directed to the `display` frame. String values retrieved from text fields are converted to number with the `parseInt()` global function.

Listing 16-35: **ScrollBy Controller**

```
<HTML>
<HEAD>
<TITLE>ScrollBy Controller</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function page(direction) {
    var pixFrame = parent.display
    var deltaY = (pixFrame.innerHeight) ? pixFrame.innerHeight :
        pixFrame.document.body.scrollHeight
    if (direction == "up") {
        deltaY = -deltaY
    }
    parent.display.scrollBy(0, deltaY)
}
function customScroll(form) {
    parent.display.scrollBy(parseInt(form.x.value), parseInt(form.y.value))
}
</SCRIPT>
</HEAD>
<BODY>
<B>ScrollBy Controller</B>
<FORM NAME="custom">
Enter an Horizontal increment
```

```
<INPUT TYPE="text" NAME="x" VALUE="0" SIZE=4">
and Vertical
<INPUT TYPE="text" NAME="y" VALUE="0" SIZE=4">
value.<BR>Then
<INPUT TYPE="button" VALUE="click to scrollBy()"
onClick="customScroll(this.form)">
<HR>
<INPUT TYPE="button" VALUE="PageDown" onClick="page('down')">
<INPUT TYPE="button" VALUE="PageUp" onClick="page('up')">

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

setCursor("*cursorType*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) in NN6 to experiment with setting the cursor. After clicking the top text box in preparation for typing, roll the cursor to a location atop an empty spot on the page. Then enter the following statements one at a time into the top text box and press Enter/Return:

```
setCursor("wait")
setCursor("spinning"
setCursor("move")
```

After evaluating each statement, roll the cursor around the page, and notice where the cursor reverts to its normal appearance.

setInterval("*expr*", *msecDelay* [, *language*])
setInterval(*funcRef*, *msecDelay* [, *funcarg1*, ..., *funcargn*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

The demonstration of the `setInterval()` method entails a two-framed environment. The framesetting document is shown in Listing 16-36.

Listing 16-36: setInterval() Demonstration Frameset

```
<HTML>
<HEAD>
<TITLE>setInterval() Method</TITLE>
</HEAD>

<FRAMESET ROWS="50%,50%">
    <FRAME SRC="lst16-37.htm" NAME="control">
    <FRAME SRC="bofright.htm" NAME="display">
</FRAMESET>
</HTML>
```

In the top frame is a control panel with several buttons that control the automatic scrolling of the Bill of Rights text document in the bottom frame. Listing 16-37 shows the control panel document. Many functions here control the interval, scrolling jump size, and direction, and they demonstrate several aspects of applying `setInterval()`.

Notice that in the beginning the script establishes a number of global variables. Three of them are parameters that control the scrolling; the last one is for the ID value returned by the `setInterval()` method. The script needs that value to be a global value so that a separate function can halt the scrolling with the `clearInterval()` method.

All scrolling is performed by the `autoScroll()` function. For the sake of simplicity, all controlling parameters are global variables. In this application, placement of those values in global variables helps the page restart autoscrolling with the same parameters as it had when it last ran.

Listing 16-37: setInterval() Control Panel

```
<HTML>
<HEAD>
<TITLE>ScrollBy Controller</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
var scrollSpeed = 500
var scrollJump = 1
var scrollDirection = "down"
var intervalID

function autoScroll() {
    if (scrollDirection == "down") {
        scrollJump = Math.abs(scrollJump)
    } else if (scrollDirection == "up" && scrollJump > 0) {
        scrollJump = -scrollJump
    }
    parent.display.scrollBy(0, scrollJump)
    if (parent.display.pageYOffset <= 0) {
```

```
        clearInterval(intervalID)
    }
}

function reduceInterval() {
    stopScroll()
    scrollSpeed -= 200
    startScroll()
}
function increaseInterval() {
    stopScroll()
    scrollSpeed += 200
    startScroll()
}
function reduceJump() {
    scrollJump -= 2
}
function increaseJump() {
    scrollJump += 2
}
function swapDirection() {
    scrollDirection = (scrollDirection == "down") ? "up" : "down"
}
function startScroll() {
    parent.display.scrollBy(0, scrollJump)
    if (intervalID) {
        clearInterval(intervalID)
    }
    intervalID = setInterval("autoScroll()",scrollSpeed)
}
function stopScroll() {
    clearInterval(intervalID)
}
</SCRIPT>
</HEAD>
<BODY onLoad="startScroll()">
<B>AutoScroll by setInterval() Controller</B>
<FORM NAME="custom">
<INPUT TYPE="button" VALUE="Start Scrolling" onClick="startScroll()">
<INPUT TYPE="button" VALUE="Stop Scrolling" onClick="stopScroll()"><P>
<INPUT TYPE="button" VALUE="Shorter Time Interval" onClick="reduceInterval()">
<INPUT TYPE="button" VALUE="Longer Time Interval"
onClick="increaseInterval()"><P>
<INPUT TYPE="button" VALUE="Bigger Scroll Jumps" onClick="increaseJump()">
<INPUT TYPE="button" VALUE="Smaller Scroll Jumps" onClick="reduceJump()"><P>
<INPUT TYPE="button" VALUE="Change Direction" onClick="swapDirection()">

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

The `setInterval()` method is invoked inside the `startScroll()` function. This function initially "burps" the page by one `scrollJump` interval so that the test in `autoScroll()` for the page being scrolled all the way to the top doesn't halt a page from scrolling before it gets started. Notice, too, that the function checks for the existence of an interval ID. If one is there, it is cleared before the new one is set. This is crucial within the design of the example page, because repeated clicking of the Start Scrolling button triggers multiple interval timers inside the browser. Only the most recent one's ID would be stored in `intervalID`, allowing no way to clear the older ones. But this little side trip makes sure that only one interval timer is running. One of the global variables, `scrollSpeed`, is used to fill the delay parameter for `setInterval()`. To change this value on the fly, the script must stop the current interval process, change the `scrollSpeed` value, and start a new process. The intensely repetitive nature of this application is nicely handled by the `setInterval()` method.

setTimeout("*expr*", *msecDelay* [, *language*])
setTimeout(*functionRef*, *msecDelay* [, *funcarg1*, ..., *funcargn*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

When you load the HTML page in Listing 16-38, it triggers the `updateTime()` function, which displays the time (in hh:mm am/pm format) in the statusbar. Instead of showing the seconds incrementing one by one (which may be distracting to someone trying to read the page), this function alternates the last character of the display between an asterisk and nothing, like a visual "heartbeat."

Listing 16-38: **Display the Current Time**

```
<HTML>
<HEAD>
<TITLE>Status Bar Clock</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var flasher = false
// calculate current time, determine flasher state,
// and insert time into status bar every second
function updateTime() {
    var now = new Date()
    var theHour = now.getHours()
    var theMin = now.getMinutes()
    var theTime = "" + ((theHour > 12) ? theHour - 12 : theHour)
    theTime += ((theMin < 10) ? ":0" : ":") + theMin
```

```
    theTime  += (theHour >= 12) ? " pm" : " am"
    theTime += ((flasher) ? " " : "*")
    flasher = !flasher
    window.status = theTime
    // recursively call this function every second to keep timer going
    timerID = setTimeout("updateTime()",1000)
}
//-->
</SCRIPT>
</HEAD>

<BODY onLoad="updateTime()">
</BODY>
</HTML>
```

In this function, the `setTimeout()` method works in the following way: Once the current time (including the flasher status) appears in the statusbar, the function waits approximately one second (1,000 milliseconds) before calling the same function again. You don't have to clear the `timerID` value in this application because JavaScript does it for you every time the 1,000 milliseconds elapse.

A logical question to ask is whether this application should be using `setInterval()` instead of `setTimeout()`. This is a case in which either one does the job. To use `setInterval()` here would require that the interval process start outside of the `updateTime()` function, because you need only one process running that repeatedly calls `updateTime()`. It would be a cleaner implementation in that regard, instead of the tons of timeout processes spawned by Listing 16-38. On the other hand, the application would not run in any browsers before NN4 or IE4, as Listing 16-38 does.

To demonstrate passing parameters, you can modify the `updateTime()` function to add the number of times it gets invoked to the display in the statusbar. For that to work, the function must have a parameter variable so that it can catch a new value each time it is invoked by `setTimeout()`'s expression. For all browsers, the function would be modified as follows (unchanged lines are represented by the ellipsis):

```
function updateTime(i) {
    ...
    window.status = theTime + "  (" + i + ")"
    // pass updated counter value with next call to this function
    timerID = setTimeout("updateTime(" + i+1 + ")",1000)
}
```

If you were running this exclusively in NN4+, you could use its more convenient way of passing parameters to the function:

```
timerID = setTimeout(updateTime,1000, i+1)
```

In either case, the `onLoad` event handler would also have to be modified to get the ball rolling with an initial parameter:

```
onLoad = "updateTime(0)"
```

One warning about `setTimeout()` functions that dive into themselves as frequently as this one does: Each call eats up a bit more memory for the browser application in Navigator 2. If you let this clock run for a while, some browsers may encounter memory difficulties, depending on which operating system they're using. But considering the amount of time the typical user spends on Web pages (even if only 10 or 15 minutes), the function shouldn't present a problem. And any reloading invoked by the user (such as by resizing the window in Navigator 2) frees up memory once again.

```
showModalDialog("URL"[, arguments]
[, features])
showModelessDialog("URL"[, arguments]
[, features])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							(✓)	✓	✓

Example

To demonstrate the two styles of dialog boxes, I have implemented the same functionality (setting some session visual preferences) for both modal and modeless dialog boxes. This tactic shows you how to pass data back and forth between the main page and both styles of dialog box windows.

The first example demonstrates how to use a modal dialog box. In the process, data is passed into the dialog box window and values are returned. Listing 16-39 is the HTML and scripting for the main page. A button's `onClick` event handler invokes a function that opens the modal dialog box. The dialog box's document (Listing 16-40) contains several form elements for entering a user name and selecting a few color styles for the main page. Data from the dialog is fashioned into an array to be sent back to the main window. That array is initially assigned to a local variable, `prefs`, as the dialog box closes. If the user cancels the dialog box, the returned value is an empty string, so nothing more in `getPrefsData()` executes. But when the user clicks OK, the array comes back. Each of the array items is read and assigned to its respective form value or style property. These values are also preserved in the global `currPrefs` array. This allows the settings to be sent to the modal dialog box (as the second parameter to `showModalDialog()`) the next time the dialog box is opened.

Listing 16-39: **Main Page for showModalDialog()**

```
<HTML>
<HEAD>
<TITLE>window.setModalDialog() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var currPrefs = new Array()
```

```
function getPrefsData() {
    var prefs = showModalDialog("lst16-40.htm", currPrefs,
        "dialogWidth:400px; dialogHeight:300px")
    if (prefs) {
        if (prefs["name"]) {
            document.all.firstName.innerText = prefs["name"]
            currPrefs["name"] = prefs["name"]
        }
        if (prefs["bgColor"]) {
            document.body.style.backgroundColor = prefs["bgColor"]
            currPrefs["bgColor"] = prefs["bgColor"]
        }
        if (prefs["textColor"]) {
            document.body.style.color = prefs["textColor"]
            currPrefs["textColor"] = prefs["textColor"]
        }
        if (prefs["h1Size"]) {
            document.all.welcomeHeader.style.fontSize = prefs["h1Size"]
            currPrefs["h1Size"] = prefs["h1Size"]
        }
    }
}
function init() {
    document.all.firstName.innerText = "friend"
}
</SCRIPT>

</HEAD>
<BODY BGCOLOR="#eeeeee" STYLE="margin:20px" onLoad="init()">
<H1>window.setModalDialog() Method</H1>
<HR>
<H2 ID="welcomeHeader">Welcome, <SPAN ID="firstName"> </SPAN>!</H2>
<HR>
<P>Use this button to set style preferences for this page:
<BUTTON ID="prefsButton" onClick="getPrefsData()">
Preferences
</BUTTON>
</BODY>
</HTML>
```

The dialog box's document, shown in Listing 16-40, is responsible for reading the incoming data (and setting the form elements accordingly) and assembling form data for return to the main window's script. Notice when you load the example that the TITLE element of the dialog box's document appears in the dialog box window's title bar.

When the page loads into the dialog box window, the `init()` function examines the `window.dialogArguments` property. If it has any data, the data is used to preset the form elements to mirror the current settings of the main page. A utility function, `setSelected()`, pre-selects the option of a SELECT element to match the current settings.

Buttons at the bottom of the page are explicitly positioned to be at the lower-right corner of the window. Each button invokes a function to do what is needed to close the dialog box. In the case of the OK button, the `handleOK()` function sets the `window.returnValue` property to the data that come back from the `getFormData()` function. This latter function reads the form element values and packages them in an array using the form elements' names as array indices. This helps keep everything straight back in the main window's script, which uses the index names, and is therefore not dependent upon the precise sequence of the form elements in the dialog box window.

Listing 16-40: **Document for the Modal Dialog**

```
<HTML>
<HEAD>
<TITLE>User Preferences</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// Close the dialog
function closeme() {
    window.close()
}

// Handle click of OK button
function handleOK() {
    window.returnValue = getFormData()
    closeme()
}

// Handle click of Cancel button
function handleCancel() {
    window.returnValue = ""
    closeme()
}
// Generic function converts form element name-value pairs
// into an array
function getFormData() {
    var form = document.prefs
    var returnedData = new Array()
    // Harvest values for each type of form element
    for (var i = 0; i < form.elements.length; i++) {
        if (form.elements[i].type == "text") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else if (form.elements[i].type.indexOf("select") != -1) {
            returnedData[form.elements[i].name] =
            form.elements[i].options[form.elements[i].selectedIndex].value
        } else if (form.elements[i].type == "radio") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else if (form.elements[i].type == "checkbox") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else continue
    }
    return returnedData
}
```

```
// Initialize by setting form elements from passed data
function init() {
    if (window.dialogArguments) {
        var args = window.dialogArguments
        var form = document.prefs
        if (args["name"]) {
            form.name.value = args["name"]
        }
        if (args["bgColor"]) {
            setSelected(form.bgColor, args["bgColor"])
        }
        if (args["textColor"]) {
            setSelected(form.textColor, args["textColor"])
        }
        if (args["h1Size"]) {
            setSelected(form.h1Size, args["h1Size"])
        }
    }
}
// Utility function to set a SELECT element to one value
function setSelected(select, value) {
    for (var i = 0; i < select.options.length; i++) {
        if (select.options[i].value == value) {
            select.selectedIndex = i
            break
        }
    }
    return
}
// Utility function to accept a press of the
// Enter key in the text field as a click of OK
function checkEnter() {
    if (window.event.keyCode == 13) {
        handleOK()
    }
}
</SCRIPT>
</HEAD>

<BODY BGCOLOR="#eeeeee" onLoad="init()">
<H2>Web Site Preferences</H2>
<HR>
<TABLE BORDER=0 CELLSPACING=2>
<FORM NAME="prefs" onSubmit="return false">
<TR>
<TD>Enter your first name:<INPUT NAME="name" TYPE="text" VALUE="" SIZE=20
onKeyDown="checkEnter()">
</TR>

<TR>
<TD>Select a background color:
<SELECT NAME="bgColor">
    <OPTION VALUE="beige">Beige
    <OPTION VALUE="antiquewhite">Antique White
```

Continued

Listing 16-40 *(continued)*

```
    <OPTION VALUE="goldenrod">Goldenrod
    <OPTION VALUE="lime">Lime
    <OPTION VALUE="powderblue">Powder Blue
    <OPTION VALUE="slategray">Slate Gray
</SELECT>
</TR>

<TR>
<TD>Select a text color:
<SELECT NAME="textColor">
    <OPTION VALUE="black">Black
    <OPTION VALUE="white">White
    <OPTION VALUE="navy">Navy Blue
    <OPTION VALUE="darkorange">Dark Orange
    <OPTION VALUE="seagreen">Sea Green
    <OPTION VALUE="teal">Teal
</SELECT>
</TR>

<TR>
<TD>Select "Welcome" heading font point size:
<SELECT NAME="h1Size">
    <OPTION VALUE="12">12
    <OPTION VALUE="14">14
    <OPTION VALUE="18">18
    <OPTION VALUE="24">24
    <OPTION VALUE="32">32
    <OPTION VALUE="48">48
</SELECT>
</TR>
</TABLE>
</FORM>
<DIV STYLE="position:absolute; left:200px; top:220px">
<BUTTON STYLE="width:80px" onClick="handleOK()">OK</BUTTON>  
<BUTTON STYLE="width:80px" onClick="handleCancel()">Cancel</BUTTON>
</DIV>
</BODY>
</HTML>
```

One last convenience feature of the dialog box window is the `onKeyPress` event handler in the text box. The function it invokes looks for the Enter key. If that key is pressed while the box has focus, the same `handleOK()` function is invoked, as if the user had clicked the OK button. This feature makes the dialog box behave as if the OK button is an automatic default, just as "real" dialog boxes.

You should observe several important structural changes that were made to turn the modal approach into a modeless one. Listing 16-41 shows the version of the main window modified for use with a modeless dialog box. Another global variable, `prefsDlog`, is initialized to eventually store the reference to the modeless window

returned by the `showModelessWindow()` method. The variable gets used to invoke the `init()` function inside the modeless dialog box, but also as conditions in an `if` construction surrounding the generation of the dialog box. The reason this is needed is to prevent multiple instances of the dialog box being created (the button is still alive while the modeless window is showing). The dialog box won't be created again as long as there is a value in `prefsDlog`, and the dialog box window has not been closed (picking up the `window.closed` property of the dialog box window).

The `showModelessDialog()` method's second parameter is a reference to the function in the main window that updates the main document. As you see in a moment, that function is invoked from the dialog box when the user clicks the OK or Apply buttons.

Listing 16-41: **Main Page for showModelessDialog()**

```
<HTML>
<HEAD>
<TITLE>window.setModelessDialog() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var currPrefs = new Array()
var prefsDlog
function getPrefsData() {
    if (!prefsDlog || prefsDlog.closed) {
        prefsDlog = showModelessDialog("lst16-42.htm", setPrefs,
        "dialogWidth:400px; dialogHeight:300px")
        prefsDlog.init(currPrefs)
    }
}

function setPrefs(prefs) {
    if (prefs["bgColor"]) {
        document.body.style.backgroundColor = prefs["bgColor"]
        currPrefs["bgColor"] = prefs["bgColor"]
    }
    if (prefs["textColor"]) {
        document.body.style.color = prefs["textColor"]
        currPrefs["textColor"] = prefs["textColor"]
    }
    if (prefs["h1Size"]) {
        document.all.welcomeHeader.style.fontSize = prefs["h1Size"]
        currPrefs["h1Size"] = prefs["h1Size"]
    }
    if (prefs["name"]) {
        document.all.firstName.innerText = prefs["name"]
        currPrefs["name"] = prefs["name"]
    }
}

function init() {
    document.all.firstName.innerText = "friend"
}
</SCRIPT>
```

Continued

Listing 16-41 *(continued)*

```
</HEAD>
<BODY BGCOLOR="#eeeeee" STYLE="margin:20px" onLoad="init()">
<H1>window.setModelessDialog() Method</H1>
<HR>
<H2 ID="welcomeHeader">Welcome, <SPAN ID="firstName"> </SPAN>!</H2>
<HR>
<P>Use this button to set style preferences for this page:
<BUTTON ID="prefsButton" onClick="getPrefsData()">
Preferences
</BUTTON>
</BODY>
</HTML>
```

Changes to the dialog box window document for a modeless version (Listing 16-42) are rather limited. A new button is added to the bottom of the screen for an Apply button. As in many dialog box windows you see in Microsoft products, the Apply button lets current settings in dialog boxes be applied to the current document but without closing the dialog box. This approach makes experimenting with settings easier.

The Apply button invokes a `handleApply()` function, which works the same as `handleOK()`, except the dialog box is not closed. But these two functions communicate back to the main window differently than a modal dialog box. The main window's processing function is passed as the second parameter of `showModelessDialog()` and is available as the `window.dialogArguments` property in the dialog box window's script. That function reference is assigned to a local variable in both functions, and the remote function is invoked, passing the results of the `getFormData()` function as parameter values back to the main window.

Listing 16-42: **Document for the Modeless Dialog Box**

```
<HTML>
<HEAD>
<TITLE>User Preferences</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// Close the dialog
function closeme() {
    window.close()
}

// Handle click of OK button
function handleOK() {
    var returnFunc = window.dialogArguments
    returnFunc(getFormData())
    closeme()
}
```

```
// Handle click of Apply button
function handleApply() {
    var returnFunc = window.dialogArguments
    returnFunc(getFormData())
}

// Handle click of Cancel button
function handleCancel() {
    window.returnValue = ""
    closeme()
}
// Generic function converts form element name-value pairs
// into an array
function getFormData() {
    var form = document.prefs
    var returnedData = new Array()
    // Harvest values for each type of form element
    for (var i = 0; i < form.elements.length; i++) {
        if (form.elements[i].type == "text") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else if (form.elements[i].type.indexOf("select") != -1) {
            returnedData[form.elements[i].name] =
            form.elements[i].options[form.elements[i].selectedIndex].value
        } else if (form.elements[i].type == "radio") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else if (form.elements[i].type == "checkbox") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else continue
    }
    return returnedData
}
// Initialize by setting form elements from passed data
function init(currPrefs) {
    if (currPrefs) {
        var form = document.prefs
        if (currPrefs["name"]) {
            form.name.value = currPrefs["name"]
        }
        if (currPrefs["bgColor"]) {
            setSelected(form.bgColor, currPrefs["bgColor"])
        }
        if (currPrefs["textColor"]) {
            setSelected(form.textColor, currPrefs["textColor"])
        }
        if (currPrefs["h1Size"]) {
            setSelected(form.h1Size, currPrefs["h1Size"])
        }
    }
}
// Utility function to set a SELECT element to one value
function setSelected(select, value) {
    for (var i = 0; i < select.options.length; i++) {
        if (select.options[i].value == value) {
```

Continued

Listing 16-42 *(continued)*

```
            select.selectedIndex = i
            break
        }
    }
    return
}
// Utility function to accept a press of the
// Enter key in the text field as a click of OK
function checkEnter() {
    if (window.event.keyCode == 13) {
        handleOK()
    }
}
</SCRIPT>
</HEAD>

<BODY BGCOLOR="#eeeeee" onLoad="init()">
<H2>Web Site Preferences</H2>
<HR>
<TABLE BORDER=0 CELLSPACING=2>
<FORM NAME="prefs" onSubmit="return false">
<TR>
<TD>Enter your first name:<INPUT NAME="name" TYPE="text" VALUE="" SIZE=20
onKeyDown="checkEnter()">
</TR>

<TR>
<TD>Select a background color:
<SELECT NAME="bgColor">
    <OPTION VALUE="beige">Beige
    <OPTION VALUE="antiquewhite">Antique White
    <OPTION VALUE="goldenrod">Goldenrod
    <OPTION VALUE="lime">Lime
    <OPTION VALUE="powderblue">Powder Blue
    <OPTION VALUE="slategray">Slate Gray
</SELECT>
</TR>

<TR>
<TD>Select a text color:
<SELECT NAME="textColor">
    <OPTION VALUE="black">Black
    <OPTION VALUE="white">White
    <OPTION VALUE="navy">Navy Blue
    <OPTION VALUE="darkorange">Dark Orange
    <OPTION VALUE="seagreen">Sea Green
    <OPTION VALUE="teal">Teal
</SELECT>
</TR>
```

```
<TR>
<TD>Select "Welcome" heading font point size:
<SELECT NAME="h1Size">
    <OPTION VALUE="12">12
    <OPTION VALUE="14">14
    <OPTION VALUE="18">18
    <OPTION VALUE="24">24
    <OPTION VALUE="32">32
    <OPTION VALUE="48">48
</SELECT>
</TR>
</TABLE>
</FORM>
<DIV STYLE="position:absolute; left:120px; top:220px">
<BUTTON STYLE="width:80px" onClick="handleOK()">OK</BUTTON>  
<BUTTON STYLE="width:80px" onClick="handleCancel()">Cancel</BUTTON>  
<BUTTON STYLE="width:80px" onClick="handleApply()">Apply</BUTTON>
</DIV>
</BODY>
</HTML>
```

The biggest design challenge you probably face with respect to these windows is deciding between a modal and modeless dialog box style. Some designers insist that modality has no place in a graphical user interface; others say that there are times when you need to focus the user on a very specific task before any further processing can take place. That's where a modal dialog box makes perfect sense.

sizeToContent()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) in NN6 to try the `sizeToContent()` method. Assuming that you are running The Evaluator from the `Chap13` directory on the CD-ROM (or the directory copied as-is to your hard disk), you can open a subwindow with one of the other files in the directory, and then size the subwindow. Enter the following statements into the top text box:

```
a = window.open("lst13-02.htm","")
a.sizeToContent()
```

The resized subwindow is at the minimum recommended width for a browser window, and at a height tall enough to display the little bit of content in the document.

Event handlers

onAfterPrint
onBeforePrint

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

The following script fragment assumes that the page includes a DIV element whose style sheet includes a setting of `display:none` as the page loads. Somewhere in the Head, the print-related event handlers are set as properties:

```
function showPrintCopyright() {
    document.all.printCopyright.style.display = "block"
}
function hidePrintCopyright() {
    document.all.printCopyright.style.display = "none"
}
window.onbeforeprint = showPrintCopyright
window.onafterprint = hidePrintCopyright
```

onBeforeUnload

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The simple page in Listing 16-43 shows you how to give the user a chance to stay on the page.

Listing 16-43: Using the onBeforeUnload Event Handler

```
<HTML>
<HEAD>
<TITLE>onBeforeUnload Event Handler</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function verifyClose() {
    event.returnValue = "We really like you and hope you will stay longer."
}
```

```
window.onbeforeunload = verifyClose
</SCRIPT>

</HEAD>
<BODY>
<H1>onBeforeUnload Event Handler</H1>
<HR>
<P>Use this button to navigate to the previous page:
<BUTTON ID="go" onClick="history.back()">
Go Back
</BUTTON>
</BODY>
</HTML>
```

onHelp

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The following script fragment can be embedded in the IE5-only modeless dialog box code in Listing 16-44 to provide context-sensitive help within the dialog box. Help messages for only two of the form elements are shown here, but in a real application you add messages for the rest.

```
function showHelp() {
    switch (event.srcElement.name) {
        case "bgColor" :
            alert("Choose a color for the main window\'s background.")
            break
        case "name" :
            alert("Enter your first name for a friendly greeting.")
            break
        default :
            alert("Make preference settings for the main page styles.")
    }
    event.returnValue = false
}
window.onhelp = showHelp
```

Because this page's help focuses on form elements, the `switch` construction cases are based on the `name` properties of the form elements. For other kinds of pages, the `id` properties may be more appropriate.

FRAME Element Object

Properties

borderColor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Although you may experience problems (especially in IE5) changing the color of a single frame border, the W3C DOM syntax would look like the following if the script were inside the framesetting document:

```
document.getElementById("contentsFrame").borderColor = "red"
```

The IE-only version would be:

```
document.all["contentsFrame"].borderColor = "red"
```

These examples assume the frame name arrives to a script function as a string. If the script is executing in one of the frames of the frameset, add a reference to `parent` in the preceding statements.

contentDocument

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

A framesetting document script might be using the ID of a FRAME element to read or adjust one of the element properties, and then need to perform some action on the content of the page through its `document` object. You can get the reference to the `document` object via a statement, such as the following:

```
var doc = document.getElementById("FRAME3").contentDocument
```

Then your script can, for example, dive into a form in the document:

```
var val = doc.mainForm.entry.value
```

Document

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

While you have far easier ways to reach the `document` object of another frame (`parent.`*`otherFrameName`*`.document`), the following statement takes the long way to get there to retrieve the number of forms in the document of another frame:

```
var formCount = parent.document.all.contentsFrame.Document.forms.length
```

Using the `Document` property only truly makes sense when a function is passed a FRAME or IFRAME element object reference as a parameter, and the script must, among other things more related to those objects, access the document contained by those elements.

frameBorder

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The default value for the `frameBorder` property is `yes`. You can use this setting to create a toggle script (which, unfortunately, does not change the appearance in IE). The W3C-compatible version looks like the following:

```
function toggleFrameScroll(frameID) {
    var theFrame = document.getElementById(frameID)
    if (theFrame.frameBorder == "yes") {
        theFrame.frameBorder = "no"
    } else {
        theFrame.frameBorder    = "yes"
    }
}
```

height
width

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The following fragment assumes a frameset defined with two frames set up as two columns within the frameset. The statements here live in the framesetting document. They retrieve the current width of the left frame and increase the width of that frame by ten percent. Syntax shown here is for the W3C DOM, but can be easily adapted to IE-only terminology.

```
var frameWidth = document.getElementById("leftFrame").width
document.getElementById("mainFrameset").cols = (Math.round(frameWidth * 1.1)) +
",*"
```

Notice how the numeric value of the existing frame width is first increased by ten percent and then concatenated to the rest of the string property assigned to the frameset's `cols` property. The asterisk after the comma means that the browser should figure out the remaining width and assign it to the right-hand frame.

noResize

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statement turns off the ability for a frame to be resized:

```
parent.document.getElementById("myFrame1").noResize = true
```

Because of the negative nature of the property name, it may be difficult to keep the logic straight (setting `noResize` to `true` means that resizability is turned off). Keep a watchful eye on your Boolean values.

scrolling

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listing 16-45 produces a frameset consisting of eight frames. The content for the frames is generated by a script within the frameset (via the `fillFrame()` function). Event handlers in the Body of each frame invoke the `toggleFrameScroll()` function. Both ways of referencing the FRAME element object are shown, with the IE-only version commented out.

In the `toggleFrameScroll()` function, the `if` condition checks whether the property is set to something other than `no`. This allows the condition to evaluate to `true` if the property is set to either `auto` (the first time) or `yes` (as set by the function). Note that the scrollbars don't disappear from the frames in IE5.5 or NN6.

Listing 16-45: **Controlling the FRAME.scrolling Property**

```
<HTML>
<HEAD>
<TITLE>frame.scrolling Property</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
function toggleFrameScroll(frameID) {
    // IE5 & NN6 version
    var theFrame = document.getElementById(frameID)
    // IE4+ version
    // var theFrame = document.all[frameID]

    if (theFrame.scrolling != "no") {
        theFrame.scrolling = "no"
    } else {
        theFrame.scrolling = "yes"
    }
}
// generate content for each frame
function fillFrame(frameID) {
    var page = "<HTML><BODY onClick='parent.toggleFrameScroll(\"" +
        frameID + "\")'><SPAN STYLE='font-size:24pt'>"
    page += "<P>This frame has the ID of:</P><P>" + frameID + ".</P>"
    page += "</SPAN></BODY></HTML>"
    return page
}
</SCRIPT>
<FRAMESET ID="outerFrameset" COLS="50%,50%">
    <FRAMESET ID="innerFrameset1" ROWS="25%,25%,25%,25%">
        <FRAME ID="myFrame1" SRC="javascript:parent.fillFrame('myFrame1')">
        <FRAME ID="myFrame2" SRC="javascript:parent.fillFrame('myFrame2')">
        <FRAME ID="myFrame3" SRC="javascript:parent.fillFrame('myFrame3')">
        <FRAME ID="myFrame4" SRC="javascript:parent.fillFrame('myFrame4')">
    </FRAMESET>
    <FRAMESET ID="innerFrameset2" ROWS="25%,25%,25%,25%">
        <FRAME ID="myFrame5" SRC="javascript:parent.fillFrame('myFrame5')">
        <FRAME ID="myFrame6" SRC="javascript:parent.fillFrame('myFrame6')">
        <FRAME ID="myFrame7" SRC="javascript:parent.fillFrame('myFrame7')">
        <FRAME ID="myFrame8" SRC="javascript:parent.fillFrame('myFrame8')">
    </FRAMESET>
</FRAMESET>
</HTML>
```

src

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

For best results, use fully formed URLs as value for the `src` property, as shown here:

```
parent.document.getElementById("mainFrame").src = "http://www.dannyg.com"
```

Relative URLs and `javascript:` pseudo-URLs will also work most of the time.

FRAMESET Element Object

Properties

border

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Even though the property is read/write in IE4+, changing the value does not change the thickness of the border you see in the browser. If you need to find the thickness of the border, a script reference from one of the frame's documents would look like the following:

```
var thickness = parent.document.all.outerFrameset.border
```

borderColor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

To retrieve the current color setting in a frameset, a script reference from one of the frame's documents would look like the following:

```
var borderColor = parent.document.all.outerFrameset.borderColor
```

cols
rows

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listings 16-46 through 16-48 show the HTML for a frameset and two of the three documents that go into the frameset. The final document is an HTML version of the U.S. Bill of Rights, which is serving here as a content frame for the demonstration.

The frameset listing (16-46) shows a three-frame setup. Down the left column is a table of contents (16-47). The right column is divided into two rows. In the top row is a simple control (16-48) that hides and shows the table of contents frame. As the user clicks the hot text of the control (located inside a SPAN element), the `onClick` event handler invokes the `toggleTOC()` function in the frameset. Figure 2-10 shows the frameset with the menu exposed.

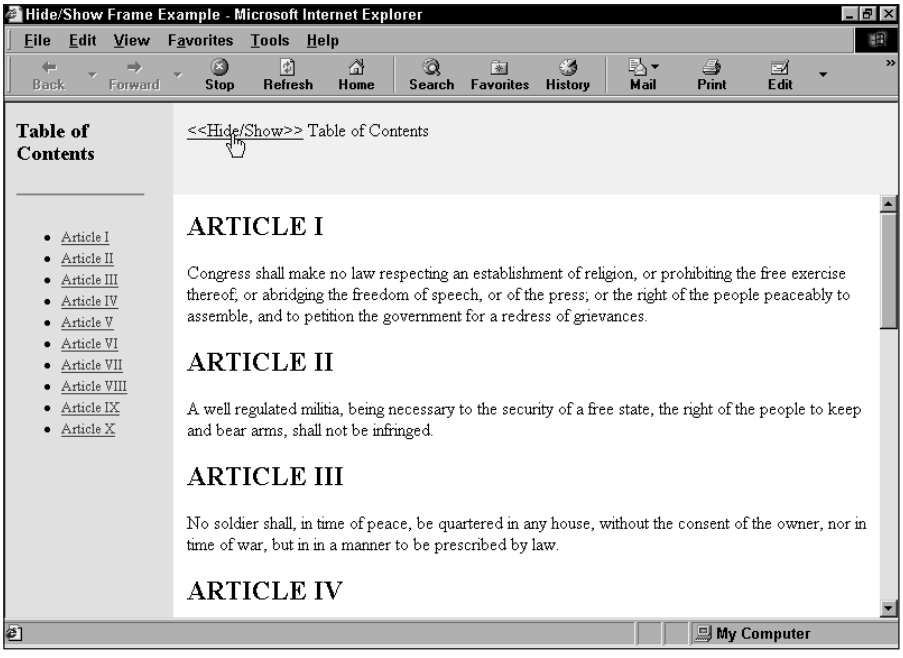


Figure 2-10: Frameset specifications are modified on the fly when you click on the top control link.

Syntax used in this example is W3C-compatible. To modify this for IE-only, you replace `document.getElementById("outerFrameset")` with `document.all.outerFrameset` and `elem.firstChild.nodeValue` to `elem.innerText`. You can also branch within the scripts to accommodate both styles.

Listing 16-46: Frameset and Script for Hiding/Showing a Frame

```
<HTML>
<HEAD>
<TITLE>Hide/Show Frame Example</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
var origCols
function toggleTOC(elem, frm) {
    if (origCols) {
        showTOC(elem)
    } else {
        hideTOC(elem, frm)
    }
}
function hideTOC(elem, frm) {
    var frameset = document.getElementById("outerFrameset")
    origCols = frameset.cols
    frameset.cols = "0,*"
}
function showTOC(elem) {
    if (origCols) {
        document.getElementById("outerFrameset").cols = origCols
        origCols = null
    }
}
</SCRIPT>
<FRAMESET ID="outerFrameset" FRAMEBORDER="no" COLS="150,*">
    <FRAME ID="TOC" NAME="TOCFrame" SRC="lst16-47.htm">
    <FRAMESET ID="innerFrameset1" ROWS="80,*">
        <FRAME ID="controls" NAME="controlsFrame" SRC="lst16-48.htm">
        <FRAME ID="content" NAME="contentFrame" SRC="bofright.htm">
    </FRAMESET>
</FRAMESET>
</HTML>
```

When a user clicks the hot spot to hide the frame, the script copies the original `cols` property settings to a global variable. The variable is used in `showTOC()` to restore the frameset to its original proportions. This allows a designer to modify the HTML for the frameset without also having to dig into scripts to hard-wire the restored size.

Listing 16-47: **Table of Contents Frame Content**

```
<HTML>
<HEAD>
<TITLE>Table of Contents</TITLE>
</HEAD>
<BODY BGCOLOR="#eeeeee">
<H3>Table of Contents</H3>
<HR>
<UL STYLE="font-size:10pt">
<LI><A HREF="bofright.htm#article1" TARGET="contentFrame">Article I</A></LI>
<LI><A HREF="bofright.htm#article2" TARGET="contentFrame">Article II</A></LI>
<LI><A HREF="bofright.htm#article3" TARGET="contentFrame">Article III</A></LI>
<LI><A HREF="bofright.htm#article4" TARGET="contentFrame">Article IV</A></LI>
<LI><A HREF="bofright.htm#article5" TARGET="contentFrame">Article V</A></LI>
<LI><A HREF="bofright.htm#article6" TARGET="contentFrame">Article VI</A></LI>
<LI><A HREF="bofright.htm#article7" TARGET="contentFrame">Article VII</A></LI>
<LI><A HREF="bofright.htm#article8" TARGET="contentFrame">Article VIII</A></LI>
<LI><A HREF="bofright.htm#article9" TARGET="contentFrame">Article IX</A></LI>
<LI><A HREF="bofright.htm#article10" TARGET="contentFrame">Article X</A></LI>
</UL>
</BODY>
</HTML>
```

Listing 16-48: **Control Panel Frame**

```
<HTML>
<HEAD>
<TITLE>Control Panel</TITLE>
</HEAD>
<BODY>
<P>
<SPAN ID="tocToggle"
      STYLE="text-decoration:underline; cursor:hand"
      onClick="parent.toggleTOC(this)"> &lt;&lt;Hide/Show&gt;&gt;</SPAN>
Table of Contents
</P>
</BODY>
</HTML>
```

frameBorder

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The default value for the `frameBorder` property is `yes`. You can use this setting to create a toggle script (which, unfortunately, does not change the appearance in IE). The IE4+-compatible version looks like the following:

```
function toggleFrameScroll(framesetID) {
    var theFrameset = document.all(framesetID)
    if (theFrameset.frameBorder == "yes") {
        theFrameset.frameBorder = "no"
    } else {
        theFrameset.frameBorder = "yes"
    }
}
```

frameSpacing

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Even though the property is read/write in IE4+, changing the value does not change the thickness of the frame spacing you see in the browser. If you need to find the spacing as set by the tag's attribute, a script reference from one of the frame's documents would look like the following:

```
var spacing = parent.document.all.outerFrameset.frameSpacing
```

IFRAME Element Object

Properties

align

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The default setting for an IFRAME alignment is `baseline`. A script can shift the IFRAME to be flush with the right edge of the containing element as follows:

```
document.getElementById("iframe1").align = "right"
```

contentDocument

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

A document script might be using the ID of an IFRAME element to read or adjust one of the element properties; it then needs to perform some action on the content of the page through its `document` object. You can get the reference to the `document` object via a statement, such as the following:

```
var doc = document.getElementById("FRAME3").contentDocument
```

Then your script can, for example, dive into a form in the document:

```
var val = doc.mainForm.entry.value
```

frameBorder

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See the example for the `FRAME.frameBorder` property earlier in this chapter.

hspace
vspace

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The following fragment sets the white space surrounding an IFRAME element to an equal amount:

```
document.all.myIframe.hspace = 20
document.all.myIframe.vspace = 20
```

Unfortunately these changes do not work for IE5/Windows.

scrolling

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following `toggleIFrameScroll()` function accepts a string of the IFRAME element's ID as a parameter and switches between on and off scroll bars in the IFRAME. The `if` condition checks whether the property is set to something other than `no`. This test allows the condition to evaluate to true `if` the property is set to either `auto` (the first time) or `yes` (as set by the function).

```
function toggleFrameScroll(frameID) {
    // IE5 & NN6 version
    var theFrame = document.getElementById(frameID)
    // IE4+ version
    // var theFrame = document.all[frameID]
    if (theFrame.scrolling != "no") {
        theFrame.scrolling = "no"
    } else {
        theFrame.scrolling = "yes"
    }
}
```

src

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

For best results, use fully formed URLs as value for the `src` property, as shown here:

```
document.getElementById("myIframe").src = "http://www.dannyg.com"
```

Relative URLs and `javascript:` pseudo-URLs also work most of the time.

popup Object

Properties

document

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `popup` object and its properties. Enter the following statements into the top text box. The first statement creates a pop-up window, whose reference is assigned to the `a` global variable. Next, a reference to the body of the pop-up's document is preserved in the `b` variable for the sake of convenience. Further statements work with these two variables.

```
a = window.createPopup()
b = a.document.body
b.style.border = "solid 2px black"
b.style.padding = "5px"
b.innerHTML = "<P>Here is some text in a popup window</P>"
a.show(200,100, 200, 50, document.body)
```

See the description of the `show()` method for details on the parameters.

isOpen

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `isOpen` property. Enter the following statements into the top text box. The sequence begins with a creation of a simple pop-up window, whose reference is assigned to the `a` global variable. Note that the final statement is actually two statements, which are designed so that the second statement executes while the pop-up window is still open.

```
a = window.createPopup()
a.document.body.innerHTML = "<P>Here is a popup window</P>"
a.show(200,100, 200, 50, document.body); alert("Popup is open:" + a.isOpen)
```

If you then click into the main window to hide the pop-up, you will see a different result if you enter the following statement into the top text box by itself:

```
alert("Popup is open:" + a.isOpen)
```

Methods

hide()
show(*left, top, width, height*[, *positioningElementRef*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Listing 16-49 demonstrates both the `show()` and `hide()` methods for a `popup` object. A click of the button on the page invokes the `selfTimer()` function, which acts as the main routine for this page. The goal is to produce a pop-up window that "self-destructs" five seconds after it appears. Along the way, a message in the pop-up counts down the seconds.

A reference to the pop-up window is preserved as a global variable, called `popup`. After the `popup` object is created, the `initContent()` function stuffs the content into the pop-up by way of assigning `style` properties and some `innerHTML` for the body of the document that is automatically created when the pop-up is generated. A SPAN element is defined so that another function later on can modify the content of just that segment of text in the pop-up. Notice that the assignment of content to the pop-up is predicated on the pop-up window having been initialized (by virtue of the `popup` variable having a value assigned to it) and that the pop-up window is not showing. While invoking `initContent()` under any other circumstances is probably impossible, the validation of the desired conditions is good programming practice.

Back in `selfTimer()`, the `popup` object is displayed. Defining the desired size requires some trial and error to make sure the pop-up window comfortably accommodates the text that is put into the pop-up in the `initContent()` function.

With the pop-up window showing, now is the time to invoke the `countDown()` function. Before the function performs any action, it validates that the pop-up has been initialized and is still visible. If a user clicks the main window while the counter is counting down, this changes the value of the `isOpen` property to `false`, and nothing inside the `if` condition executes.

This `countDown()` function grabs the inner text of the SPAN and uses `paresInt()` to extract just the integer number (using base 10 numbering, because we're dealing with zero-leading numbers that can potentially be regarded as octal values). The condition of the `if` construction decreases the retrieved integer by one. If the decremented value is zero, then the time is up, and the pop-up window is

hidden with the `popup` global variable returned to its original, `null` value. But if the value is other than zero, then the inner text of the SPAN is set to the decremented value (with a leading zero), and the `setTimeout()` method is called upon to reinvoke the `countDown()` function in one second (1000 milliseconds).

Listing 16-49: **Hiding and Showing a Pop-up**

```
<HTML>
<HEAD>
<TITLE>popup Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var popup
function initContent() {
    if (popup && !popup.isOpen) {
        var popBody = popup.document.body
        popBody.style.border = "solid 3px red"
        popBody.style.padding = "10px"
        popBody.style.fontSize = "24pt"
        popBody.style.textAlign = "center"
        var bodyText = "<P>This popup will self-destruct in "
        bodyText += "<SPAN ID='counter'>05</SPAN>"
        bodyText += " seconds...</P>"
        popBody.innerHTML = bodyText
    }
}
function countDown() {
    if (popup && popup.isOpen) {
        var currCount = parseInt(popup.document.all.counter.innerText, 10)
        if (--currCount == 0) {
            popup.hide()
            popup = null
        } else {
            popup.document.all.counter.innerText = "0" + currCount
            setTimeout("countDown()", 1000)
        }
    }
}
function selfTimer() {
    popup = window.createPopup()
    initContent()
    popup.show(200,200,400,100,document.body)
    setTimeout("countDown()", 1000)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Impossible Mission" onClick="selfTimer()">
</FORM>
</BODY>
</HTML>
```

The `hide()` method here is invoked by a script that is running while the pop-up window is showing. Because a pop-up window automatically goes away if a user clicks the main window, it is highly unlikely that the `hide()` method would ever be invoked by itself in response to user action in the main window. If you want a script in the pop-up window to close the pop-up, use `parentWindow.close()`.

CHAPTER 2

Window and Frame Objects (Chapter 16)

In This Chapter

Scripting communication among multiple frames

Creating and managing new windows

Controlling the size, position, and appearance of the browser window

Dynamically adjusting frame sizes and frameset compositions

As physical containers of documents, window and frame objects play huge rolls in scripting. The `window` object has been scriptable in one form or another since the first scriptable browsers. Of course the object has gained numerous properties, methods, and event handlers over time, but you also often find many object-model-specific items that you probably wish were available across all browsers.

While scripts permit Web authors to manage multiple windows — and many of the examples in this chapter support that facility — try to think about your visitors, too. Very often multiple windows get in the way of site navigation and content, regardless of your good intentions. As some examples also demonstrate, you must include safety nets for your code to counteract the unpredictable actions of users who close or hide windows precisely when you don't want them to do so. Therefore, do not regard the multi-window examples here as user interface recommendations; rather consider them as recommended ways to handle a potentially tricky user-interface element.

Possible exceptions to my multi-window admonitions are the modal and modeless dialog box windows provided by various versions of IE for Windows. For other platforms, a modal dialog box can be simulated (search for details at `www.dannyg.com`). IE5.5 for Windows also adds a popup type window, which can be a helpful user interface element that exists between a tooltip and a modal dialog box.

Modern browsers, however, provide ample script control over framesets. As examples in this chapter demonstrate, your scripts can hide and show frames, or completely rearchitect a frameset without loading a new frameset.

Examples Highlights

- Listing 16-4 for the `window.closed` property demonstrates an industrial-strength treatment of new window creation, which works with all scriptable browsers (taking into account shortcomings of earlier browsers).
- NN4+ allows dynamic control over the presence of window chrome (statusbar, toolbar, et al.) with the help of signed scripts, as shown in Listing 16-6. Without signed scripts, or for IE, you must use `window.open()` to create a separate window with the characteristics of your choice.
- The example listings for the `window.opener` property show you how scripts from a subwindow communicate with the window that opened it.
- In the example listings for the `window.parent` property, you see how references to the various synonyms for a window object within a frameset evaluate. Thus, you can see what the references `window`, `top`, `parent`, and `self` mean within a frameset.
- Compare Listings 16-20, 16-23, and 16-29 to understand not only the different looks of the three native dialog box windows (alert, confirm, and prompt), but also how values returned from two of them can influence script processing sequences.
- A simple countdown timer in Listing 16-22 shows a practical application of the `window.clearTimeout()` method. Here the method stops the looping timer when the count reaches zero.
- Watch the browser window dance in Listing 16-24. The `window.moveBy()` and `window.moveTo()` methods put window positioning through its paces.
- Examples for `window.setInterval()` and `window.setTimeout()` apply these two similar methods to applications that are ideal for each one. You find other applications of `setTimeout()` in examples for the `window.closed` property and `window.open()` method.
- Internet Explorer's modal and modeless dialog box windows get workouts in Listings 16-39 through 16-42.
- The composition of a frameset, including the sizes of the frames, can be controlled dynamically in IE4+ and NN6, as shown in examples for the `FRAMESET.cols` and `FRAMESET.rows` properties.

Window Object

Properties

clipboardData

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

See Listings 15-30 and 15-39 (in Chapter 1 of this book) to see how the `clipboardData` object is used with a variety of edit-related event handlers.

closed

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

In Listing 16-4, I have created the ultimate cross-platform window opening and closing sample. It takes into account the lack of the `opener` property in Navigator 2, the missing `closed` property in Navigator 2 and Internet Explorer 3, and it even provides an ugly but necessary workaround for the inability of Internet Explorer 3 to gracefully see if a subwindow is still open.

The script begins by initializing a global variable, `newWind`, which is used to hold the object reference to the second window. This value needs to be global so that other functions can reference the window for tasks, such as closing. Another global variable, `isIE3`, is a Boolean flag that lets the window closing routines know whether the visitor is using Internet Explorer 3 (see details about the `navigator.appVersion` property in Chapter 28 of the *JavaScript Bible*).

For this example, the new window contains some HTML code written dynamically to it, rather than loading an existing HTML file into it. Therefore, the URL parameter of the `window.open()` method is left as an empty string. It is vital, however, to assign a name in the second parameter to accommodate the Internet Explorer 3 workaround for closing the window. After the new window is opened, an `opener` property is assigned to the object if one is not already assigned (this property is needed only for Navigator 2). Next comes a brief delay to allow Internet Explorer (especially versions 3 and 4) to catch up with opening the window so that content

can be written to it. The delay (using the `setTimeout()` method described later in this chapter) invokes the `finishNewWindow()` function, which uses the global `newWind` variable to reference the window for writing. The `document.close()` method closes writing to the document — a different kind of close than a window close.

A separate function, `closeWindow()`, is responsible for closing the subwindow. To accommodate Internet Explorer 3, the script appears to create another window with the same characteristics as the one opened earlier in the script. This is the trick: If the earlier window exists (with exactly the same parameters and a name *other* than an empty string), Internet Explorer does not create a new window even with the `window.open()` method executing in plain sight. To the user, nothing unusual appears on the screen. Things look weird for Internet Explorer 3 users only if the user has closed the subwindow. The `window.open()` method momentarily creates that subwindow. This subwindow is necessary because a "living" `window` object must be available for the upcoming test of window existence. (Internet Explorer 3 displays a script error if you try to address a missing window, while NN2+ and IE4+ simply return friendly `null` values.)

As a final test, an `if` condition looks at two conditions: 1) if the `window` object has ever been initialized with a value other than `null` (in case you click the window closing button before ever having created the new window) and 2) if the window's `closed` property is `null` or `false`. If either condition is true, the `close()` method is sent to the second window.

Listing 16-4: **Checking Before Closing a Window**

```
<HTML>
<HEAD>
<TITLE>window.closed Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// initialize global var for new window object
// so it can be accessed by all functions on the page
var newWind
// set flag to help out with special handling for window closing
var isIE3 = (navigator.appVersion.indexOf("MSIE 3") != -1) ? true : false
// make the new window and put some stuff in it
function newWindow() {
    newWind = window.open("","subwindow","HEIGHT=200,WIDTH=200")
    // take care of Navigator 2
    if (newWind.opener == null) {
        newWind.opener = window
    }
    setTimeout("finishNewWindow()", 100)
}
function finishNewWindow() {
    var output = ""
    output += "<HTML><BODY><H1>A Sub-window</H1>"
    output += "<FORM><INPUT TYPE='button' VALUE='Close Main Window'"
    output +="onClick='window.opener.close()'></FORM></BODY></HTML>"
```

```
    newWind.document.write(output)
    newWind.document.close()
}
// close subwindow, including ugly workaround for IE3
function closeWindow() {
    if (isIE3) {
        // if window is already open, nothing appears to happen
        // but if not, the subwindow flashes momentarily (yech!)
        newWind = window.open("","subwindow","HEIGHT=200,WIDTH=200")
    }
    if (newWind && !newWind.closed) {
        newWind.close()
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Open Window" onClick="newWindow()"><BR>
<INPUT TYPE="button" VALUE="Close it if Still Open" onClick="closeWindow()">
</FORM>
</BODY>
</HTML>
```

To complete the example of the window opening and closing, notice that the subwindow is given a button whose `onClick` event handler closes the main window. In Navigator 2 and Internet Explorer 3, this occurs without complaint. But in NN3+ and IE4+, the user is presented with an alert asking to confirm the closure of the main browser window.

defaultStatus

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Unless you plan to change the default statusbar text while a user spends time at your Web page, the best time to set the property is when the document loads. In Listing 16-5, notice how I also read this property to reset the statusbar in an `onMouseOut` event handler. Setting the `status` property to empty also resets the statusbar to the `defaultStatus` setting.

Listing 16-5: **Setting the Default Status Message**

```
<HTML>
<HEAD>
<TITLE>window.defaultStatus property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
window.defaultStatus = "Welcome to my Web site."
</SCRIPT>
</HEAD>
<BODY>
<A HREF="http://www.microsoft.com"
onMouseOver="window.status = 'Visit Microsoft\'s Home page.';return true"
onMouseOut="window.status = '';return true">Microsoft</A><P>
<A HREF="http://home.netscape.com"
onMouseOver="window.status = 'Visit Netscape\'s Home page.';return true"
onMouseOut="window.status = window.defaultStatus;return true">Netscape</A>
</BODY>
</HTML>
```

If you need to display single or double quotes in the statusbar (as in the second link in Listing 16-5), use escape characters (`\'` and `\"`) as part of the strings being assigned to these properties.

dialogArguments

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 16-38 for the `window.showModalDialog()` method to see how arguments can be passed to a dialog box and retrieved via the `dialogArguments` property.

dialogHeight
dialogWidth

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Dialog boxes sometimes provide a button or icon that reveals more details or more complex settings for advanced users. You can create a function that handles the toggle between two sizes. The following function assumes that the document in the dialog box has a button whose label also toggles between "Show Details" and "Hide Details." The button's `onClick` event handler invokes the function as `toggleDetails(this)`.

```
function toggleDetails(btn) {
    if (dialogHeight == "200px") {
        dialogHeight = "350px"
        btn.value = "Hide Details"
    } else {
        dialogHeight = "200px"
        btn.value = "Show Details"
    }
}
```

In practice, you also have to toggle the `display` style sheet property of the extra material between `none` and `block` to make sure that the dialog box does not display scrollbars in the smaller dialog box version.

dialogLeft
dialogTop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Although usually not a good idea because of the potentially jarring effect on a user, you can reposition a dialog box window that has been resized by script (or by the user if you let the dialog box be resizable). The following statements in a dialog box window document's script recenter the dialog box window.

```
dialogLeft = (screen.availWidth/2) - (parseInt(dialogWidth)/2) + "px"
dialogHeight = (screen.availHeight/2) - (parseInt(dialogHeight)/2) + "px"
```

Note that the `parseInt()` functions are used to read the numeric portion of the `dialogWidth` and `dialogHeight` properties so that the values can be used for arithmetic.

directories
locationbar
menubar
personalbar
scrollbars
statusbar
toolbar

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

In Listing 16-6, you can experiment with the look of a browser window with any of the chrome elements turned on and off. To run this script, you must either sign the scripts or turn on codebase principals (see Chapter 46 of the *JavaScript Bible*). Java must also be enabled to use the signed script statements.

As the page loads, it stores the current state of each chrome element. One button for each chrome element triggers the `toggleBar()` function. This function inverts the visible property for the chrome object passed as a parameter to the function. Finally, the Restore button returns visibility to their original settings. Notice that the `restore()` function is also called by the `onUnload` event handler for the document. Also, if you load this example into NN6, non-fatal script errors occur when the scrollbars are turned on or off.

Listing 16-6: **Controlling Window Chrome**

```
<HTML>
<HEAD>
<TITLE>Bars Bars Bars</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// store original outer dimensions as page loads
var originalLocationbar = window.locationbar.visible
var originalMenubar = window.menubar.visible
var originalPersonalbar = window.personalbar.visible
var originalScrollbars = window.scrollbars.visible
var originalStatusbar = window.statusbar.visible
var originalToolbar = window.toolbar.visible

// generic function to set inner dimensions
function toggleBar(bar) {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite")
    bar.visible = !bar.visible
    netscape.security.PrivilegeManager.revertPrivilege("UniversalBrowserWrite")
}
```

```
// restore settings
function restore() {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite")
    window.locationbar.visible = originalLocationbar
    window.menubar.visible = originalMenubar
    window.personalbar.visible = originalPersonalbar
    window.scrollbars.visible = originalScrollbars
    window.statusbar.visible = originalStatusbar
    window.toolbar.visible = originalToolbar
    netscape.security.PrivilegeManager.revertPrivilege("UniversalBrowserWrite")
}
</SCRIPT>
</HEAD>
<BODY onUnload="restore()">
<FORM>
<B>Toggle Window Bars</B><BR>
<INPUT TYPE="button" VALUE="Location Bar"
onClick="toggleBar(window.locationbar)"><BR>
<INPUT TYPE="button" VALUE="Menu Bar" onClick="toggleBar(window.menubar)"><BR>
<INPUT TYPE="button" VALUE="Personal Bar"
onClick="toggleBar(window.personalbar)"><BR>
<INPUT TYPE="button" VALUE="Scrollbars"
onClick="toggleBar(window.scrollbars)"><BR>
<INPUT TYPE="button" VALUE="Status Bar"
onClick="toggleBar(window.statusbar)"><BR>
<INPUT TYPE="button" VALUE="Tool Bar" onClick="toggleBar(window.toolbar)"><BR>
<HR>
<INPUT TYPE="button" VALUE="Restore Original Settings" onClick="restore()"><BR>
</FORM>
</BODY>
</HTML>
```

external

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The first example asks the user if it is okay to add a Web site to the Active Desktop. If Active Desktop is not enabled, the user is given the choice of enabling it at this point.

```
external.AddDesktopComponent("http://www.nytimes.com","website", 200, 100, 400, 400)
```

In the next example, the user is asked to approve the addition of a URL to the Favorites list. The user can follow the normal procedure for filing the item in a folder in the list.

```
external.AddFavorite("http://www.dannyg.com/update6.html",
"JSBible 4 Support Center")
```

The final example assumes that a user makes a choice from a SELECT list of items. The `onChange` event handler of the SELECT list invokes the following function to navigate to a fictitious page and locate listings for a chosen sports team on the page.

```
function locate(list) {
    var choice = list.options[list.selectedIndex].value
    external.NavigateAndFind("http://www.collegesports.net/scores.html", choice,
"scores")
}
```

frames

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listings 16-7 and 16-8 demonstrate how JavaScript treats values of frame references from objects inside a frame. The same document is loaded into each frame. A script in that document extracts info about the current frame and the entire frameset. Figure 2-1 shows the results after loading the HTML document in Listing 16-7.

Listing 16-7: Framesetting Document for Listing 16-8

```
<HTML>
<HEAD>
<TITLE>window.frames property</TITLE>
</HEAD>
<FRAMESET COLS="50%,50%">
    <FRAME NAME="JustAKid1" SRC="lst16-08.htm">
    <FRAME NAME="JustAKid2" SRC="lst16-08.htm">
</FRAMESET>
</HTML>
```

A call to determine the number (length) of frames returns 0 from the point of view of the current frame referenced. That's because each frame here is a window that has no nested frames within it. But add the `parent` property to the reference, and the scope zooms out to take into account all frames generated by the parent window's document.

Listing 16-8: **Showing Various Window Properties**

```
<HTML>
<HEAD>
<TITLE>Window Revealer II</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function gatherWindowData() {
    var msg = ""
    msg += "<B>From the point of view of this frame:</B><BR>"
    msg += "window.frames.length: " + window.frames.length + "<BR>"
    msg += "window.name: " + window.name + "<P>"
    msg += "<B>From the point of view of the framesetting document:</B><BR>"
    msg += "parent.frames.length: " + parent.frames.length + "<BR>"
    msg += "parent.frames[0].name: " + parent.frames[0].name
    return msg
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write(gatherWindowData())
</SCRIPT>
</BODY>
</HTML>
```

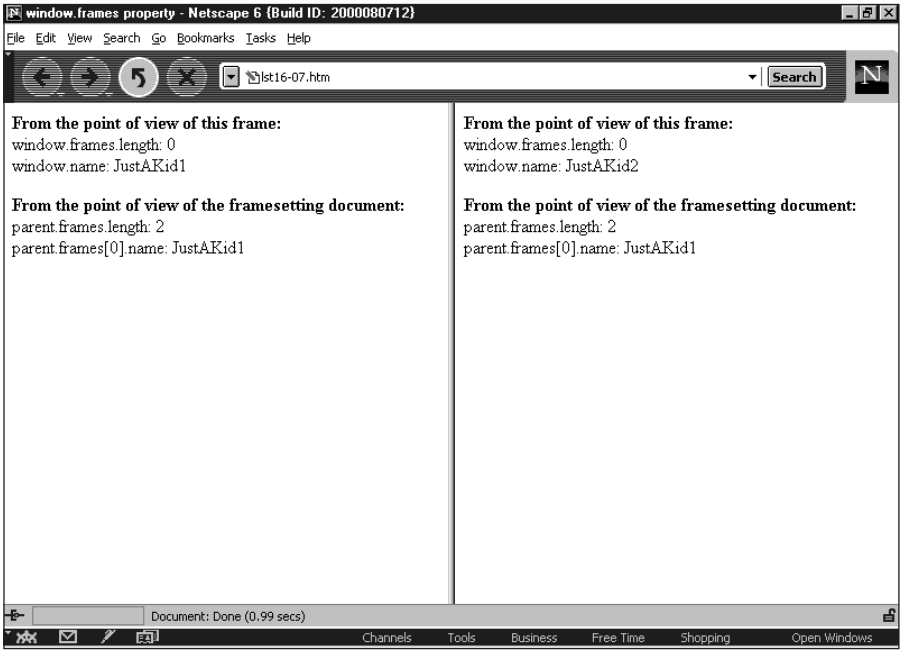


Figure 2-1: Property readouts from both frames loaded from Listing 16-7

The last statement in the example shows how to use the array syntax (brackets) to refer to a specific frame. All array indexes start with 0 for the first entry. Because the document asks for the name of the first frame (`parent.frames[0]`), the response is `JustAKid1` for both frames.

```
innerHeight
innerWidth
outerHeight
outerWidth
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

In Listing 16-9, a number of buttons let you see the results of setting the `innerHeight`, `innerWidth`, `outerHeight`, and `outerWidth` properties.

Listing 16-9: Setting Window Height and Width

```
<HTML>
<HEAD>
<TITLE>Window Sizer</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// store original outer dimensions as page loads
var originalWidth = window.outerWidth
var originalHeight = window.outerHeight
// generic function to set inner dimensions
function setInner(width, height) {
    window.innerWidth = width
    window.innerHeight = height
}
// generic function to set outer dimensions
function setOuter(width, height) {
    window.outerWidth = width
    window.outerHeight = height
}
// restore window to original dimensions
function restore() {
    window.outerWidth = originalWidth
    window.outerHeight = originalHeight
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<B>Setting Inner Sizes</B><BR>
```

```
<INPUT TYPE="button" VALUE="600 Pixels Square" onClick="setInner(600,600)"><BR>
<INPUT TYPE="button" VALUE="300 Pixels Square" onClick="setInner(300,300)"><BR>
<INPUT TYPE="button" VALUE="Available Screen Space"
onClick="setInner(screen.availWidth, screen.availHeight)"><BR>
<HR>
<B>Setting Outer Sizes</B><BR>
<INPUT TYPE="button" VALUE="600 Pixels Square" onClick="setOuter(600,600)"><BR>
<INPUT TYPE="button" VALUE="300 Pixels Square" onClick="setOuter(300,300)"><BR>
<INPUT TYPE="button" VALUE="Available Screen Space"
onClick="setOuter(screen.availWidth, screen.availHeight)"><BR>
<HR>
<INPUT TYPE="button" VALUE="Cinch up for Win95" onClick="setInner(273,304)"><BR>
<INPUT TYPE="button" VALUE="Cinch up for Mac" onClick="setInner(273,304)"><BR>
<INPUT TYPE="button" VALUE="Restore Original" onClick="restore()"><BR>
</FORM>
</BODY>
</HTML>
```

As the document loads, it saves the current outer dimensions in global variables. One of the buttons restores the windows to these settings. Two parallel sets of buttons set the inner and outer dimensions to the same pixel values so that you can see the effects on the overall window and document area when a script changes the various properties.

Because Navigator 4 displays different-looking buttons in different platforms (as well as other elements), the two buttons contain script instructions to size the window to best display the window contents. Unfortunately, no measure of the active area of a document is available, so that the dimension values were determined by trial and error before being hard-wired into the script.

navigator

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

This book is littered with examples of using the `navigator` object, primarily for performing browser detection. Examples of specific `navigator` object properties can be found in Chapter 28 of the *JavaScript Bible* and Chapter 12 of this book.

offscreenBuffering

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

If you want to turn off buffering for an entire page, include the following statement at the beginning of your script statements:

```
window.offscreenBuffering = false
```

onerror

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

In Listing 16-10, one button triggers a script that contains an error. I've added an error-handling function to process the error so that it opens a separate window and fills in a textarea form element (see Figure 2-2). If you load Listing 16-10 in NN6, some of the reporting categories report "undefined" because the browser unfortunately does not pass error properties to the `handleError()` function. A Submit button is also provided to mail the bug information to a support center e-mail address — an example of how to handle the occurrence of a bug in your scripts.

Listing 16-10: **Controlling Script Errors**

```
<HTML>
<TITLE>Error Dialog Control</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
// function with invalid variable value
function goWrong() {
    var x = fred
}
// turn off error dialogs
function errOff() {
    window.onerror = doNothing
}
// turn on error dialogs with hard reload
function errOn() {
    window.onerror = handleError
}

// assign default error handler
window.onerror = handleError

// error handler when errors are turned off...prevents error dialog
function doNothing() {return true}

function handleError(msg, URL, lineNum) {
    var errWind = window.open("","errors","HEIGHT=270,WIDTH=400")
    var wintxt = "<HTML><BODY BGCOLOR=RED>"
```

```
    wintxt += "<B>An error has occurred on this page.  "
    wintxt += "Please report it to Tech Support.</B>"
    wintxt += "<FORM METHOD=POST ENCTYPE='text/plain' "
    wintxt += "ACTION=mailTo:support4@dannyg.com >"
    wintxt += "<TEXTAREA NAME='errMsg' COLS=45 ROWS=8 WRAP=VIRTUAL>"
    wintxt += "Error: " + msg + "\n"
    wintxt += "URL: " + URL + "\n"
    wintxt += "Line: " + lineNum + "\n"
    wintxt += "Client: " + navigator.userAgent + "\n"
    wintxt += "----------------------------------------\n"
    wintxt += "Please describe what you were doing when the error occurred:"
    wintxt += "</TEXTAREA><P>"
    wintxt += "<INPUT TYPE=SUBMIT VALUE='Send Error Report'>"
    wintxt += "<INPUT TYPE=button VALUE='Close' onClick='self.close()'>"
    wintxt += "</FORM></BODY></HTML>"
    errWind.document.write(wintxt)
    errWind.document.close()
    return true
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform">
<INPUT TYPE="button" VALUE="Cause an Error" onClick="goWrong()"><P>
<INPUT TYPE="button" VALUE="Turn Off Error Dialogs" onClick="errOff()">
<INPUT TYPE="button" VALUE="Turn On Error Dialogs" onClick="errOn()">
</FORM>
</BODY>
</HTML>
```

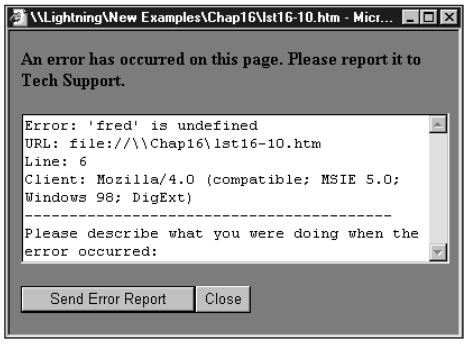


Figure 2-2: An example of a self-reporting error window

I provide a button that performs a hard reload, which, in turn, resets the `window.onerror` property to its default value. With error dialog boxes turned off, the error-handling function does not run.

opener

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓	✓	✓	✓	✓	✓

Example

To demonstrate the importance of the `opener` property, take a look at how a new window can define itself from settings in the main window (Listing 16-11). The `doNew()` function generates a small subwindow and loads the file in Listing 16-12 into the window. Notice the initial conditional statements in `doNew()` to make sure that if the new window already exists, it comes to the front by invoking the new window's `focus()` method. You can see the results in Figure 2-3. Because the `doNew()` function in Listing 16-11 uses window methods and properties not available in IE3, this example does not work correctly in IE3.

Listing 16-11: Contents of a Main Window Document That Generates a Second Window

```
<HTML>
<HEAD>
<TITLE>Master of all Windows</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
var myWind
function doNew() {
    if (!myWind || myWind.closed) {
        myWind = window.open("lst16-12.htm","subWindow",
           "HEIGHT=200,WIDTH=350,resizable")
    } else {
        // bring existing subwindow to the front
        myWind.focus()
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="input">
Select a color for a new window:
<INPUT TYPE="radio" NAME="color" VALUE="red" CHECKED>Red
<INPUT TYPE="radio" NAME="color" VALUE="yellow">Yellow
<INPUT TYPE="radio" NAME="color" VALUE="blue">Blue
<INPUT TYPE="button" NAME="storage" VALUE="Make a Window" onClick="doNew()">
<HR>
This field will be filled from an entry in another window:
<INPUT TYPE="text" NAME="entry" SIZE=25>
</FORM>
</BODY>
</HTML>
```

The `window.open()` method doesn't provide parameters for setting the new window's background color, so I let the `getColor()` function in the new window do the job as the document loads. The function uses the `opener` property to find out which radio button on the main page is selected.

Listing 16-12: References to the opener Property

```
<HTML>
<HEAD>
<TITLE>New Window on the Block</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function getColor() {
    // shorten the reference
    colorButtons = self.opener.document.forms[0].color
    // see which radio button is checked
    for (var i = 0; i < colorButtons.length; i++) {
        if (colorButtons[i].checked) {
            return colorButtons[i].value
        }
    }
    return "white"
}
</SCRIPT>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
document.write("<BODY BGCOLOR='" + getColor() + "'>")
</SCRIPT>
<H1>This is a new window.</H1>
<FORM>
<INPUT TYPE="button" VALUE="Who's in the Main window?"
onClick="alert(self.opener.document.title)"><P>
Type text here for the main window:
<INPUT TYPE="text" SIZE=25 onChange="self.opener.document.forms[0].entry.value =
this.value">
</FORM>
</BODY>
</HTML>
```

In the `getColor()` function, the multiple references to the radio button array can be very long. To simplify the references, the `getColor()` function starts out by assigning the radio button array to a variable I arbitrarily call `colorButtons`. That shorthand now stands in for lengthy references as I loop through the radio buttons to determine which button is checked and retrieve its value property.

A button in the second window simply fetches the title of the opener window's document. Even if another document loads in the main window in the meantime, the `opener` reference still points to the main window: Its `document` object, however, will change.

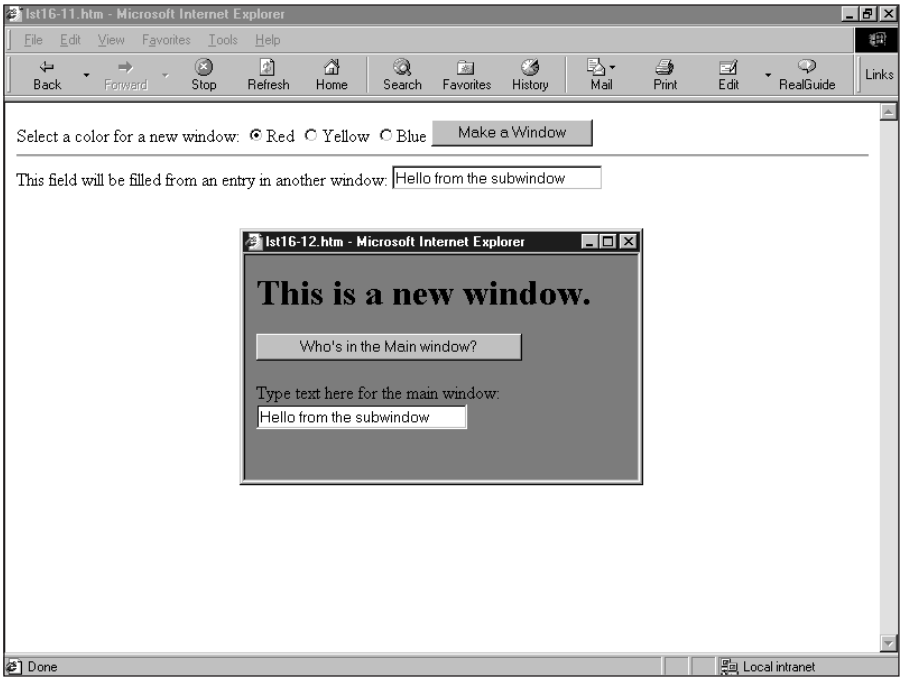


Figure 2-3: The main and subwindows, inextricably linked via the window.opener property

Finally, the second window contains a text input object. Enter any text there that you like and either tab or click out of the field. The `onChange` event handler updates the field in the opener's document (provided that document is still loaded).

pageXOffset
pageYOffset

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

The script in Listing 16-13 is an unusual construction that creates a frameset and creates the content for each of the two frames all within a single HTML document (see "Frame Object" in Chapter 16 of the *JavaScript Bible* for more details). The purpose of this example is to provide you with a playground to become familiar with the page offset concept and how the values of these properties correspond to physical activity in a scrollable document.

In the left frame of the frameset are two fields that are ready to show the pixel values of the right frame's `pageXOffset` and `pageYOffset` properties. The content

of the right frame is a 30-row table of fixed width (800 pixels). Mouse click events are captured by the document level (see Chapter 18 of the *JavaScript Bible*), allowing you to click any table or cell border or outside the table to trigger the `showOffsets()` function in the right frame. That function is a simple script that displays the page offset values in their respective fields in the left frame.

Listing 16-13: Viewing the pageXOffset and pageYOffset Properties

```
<HTML>
<HEAD>
<TITLE>Master of all Windows</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function leftFrame() {
    var output = "<HTML><BODY><H3>Page Offset Values</H3><HR>\n"
    output += "<FORM>PageXOffset:<INPUT TYPE='text' NAME='xOffset' SIZE=4><BR>\n"
    output += "PageYOffset:<INPUT TYPE='text' NAME='yOffset' SIZE=4><BR>\n"
    output += "</FORM></BODY></HTML>"
    return output
}

function rightFrame() {
    var output = "<HTML><HEAD><SCRIPT LANGUAGE='JavaScript'>\n"
    output += "function showOffsets() {\n"
    output += "parent.readout.document.forms[0].xOffset.value =
self.pageXOffset\n"
    output += "parent.readout.document.forms[0].yOffset.value =
self.pageYOffset\n}\n"
    output += "document.captureEvents(Event.CLICK)\n"
    output += "document.onclick = showOffsets\n"
    output += "<\/SCRIPT></HEAD><BODY><H3>Content Page</H3>\n"
    output += "Scroll this frame and click on a table border to view " +
              "page offset values.<BR><HR>\n"
    output += "<TABLE BORDER=5 WIDTH=800>"
    var oneRow = "<TD>Cell 1</TD><TD>Cell 2</TD><TD>Cell 3</TD>" +
                 "<TD>Cell 4</TD><TD>Cell 5</TD>"
    for (var i = 1; i <= 30; i++) {
        output += "<TR><TD><B>Row " + i + "</B></TD>" + oneRow + "</TR>"
    }
    output += "</TABLE></BODY></HTML>"
    return output
}
</SCRIPT>
</HEAD>
<FRAMESET COLS="30%,70%">
    <FRAME NAME="readout" SRC="javascript:parent.leftFrame()">
    <FRAME NAME="display" SRC="javascript:parent.rightFrame()">
</FRAMESET>
</HTML>
```

To gain an understanding of how the offset values work, scroll the window slightly in the horizontal direction and notice that the `pageXOffset` value increases; the same goes for the `pageYOffset` value as you scroll down. Remember that these values reflect the coordinate in the document that is currently under the top-left corner of the window (frame) holding the document. You can see an IE4+ version of this example in Listing 18-20 (in Chapter 4 of this book). A cross-browser version would require very little browser branching.

parent

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

To demonstrate how various `window` object properties refer to window levels in a multiframe environment, use your browser to load the Listing 16-14 document. It, in turn, sets each of two equal-size frames to the same document: Listing 16-15. This document extracts the values of several window properties, plus the `document.title` properties of two different window references.

Listing 16-14: **Framesetting Document for Listing 16-15**

```
<HTML>
<HEAD>
<TITLE>The Parent Property Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
self.name = "Framesetter"
</SCRIPT>
</HEAD>
<FRAMESET COLS="50%,50%" onUnload="self.name = ''">
    <FRAME NAME="JustAKid1" SRC="lst16-15.htm">
    <FRAME NAME="JustAKid2" SRC="lst16-15.htm">
</FRAMESET>
</HTML>
```

Listing 16-15: **Revealing Various Window-Related Properties**

```
<HTML>
<HEAD>
<TITLE>Window Revealer II</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function gatherWindowData() {
    var msg = ""
```

```
    msg = msg + "top name: " + top.name + "<BR>"
    msg = msg + "parent name: " + parent.name + "<BR>"
    msg = msg + "parent.document.title: " + parent.document.title + "<P>"
    msg = msg + "window name: " + window.name + "<BR>"
    msg = msg + "self name: " + self.name + "<BR>"
    msg = msg + "self.document.title: " + self.document.title
    return msg
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write(gatherWindowData())
</SCRIPT>
</BODY>
</HTML>
```

In the two frames (Figure 2-4), the references to the `window` and `self` object names return the name assigned to the frame by the frameset definition (`JustAKid1` for the left frame, `JustAKid2` for the right frame). In other words, from each frame's point of view, the `window` object is its own frame. References to `self.document.title` refer only to the document loaded into that window frame. But references to the top and parent windows (which are one and the same in this example) show that those object properties are shared between both frames.

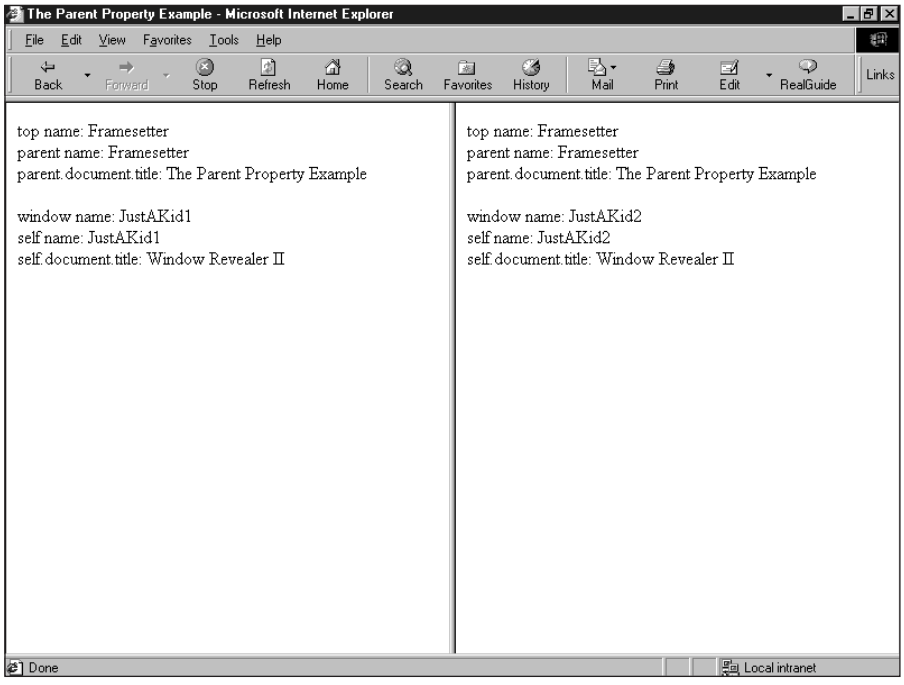


Figure 2-4: Parent and top properties being shared by both frames

A couple other fine points are worth highlighting. First, the name of the framesetting window is set as Listing 16-14 loads, rather than in response to an `onLoad` event handler in the `<FRAMESET>` tag. The reason for this is that the name must be set in time for the documents loading in the frames to get that value. If I had waited until the frameset's `onLoad` event handler, the name wouldn't be set until after the frame documents had loaded. Second, I restore the parent window's name to an empty string when the framesetting document unloads. This is to prevent future pages from getting confused about the window name.

returnValue

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 16-39 for the `showModalDialog()` method for an example of how to get data back from a dialog box in IE4+.

screenLeft
screenTop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `screenLeft` and `screenTop` properties. Start with the browser window maximized (if you are using Windows). Enter the following property name into the top text box:

```
window.screenLeft
```

Click the Evaluate button to see the current setting. Unmaximize the window and drag it around the screen. Each time you finish dragging, click the Evaluate button again to see the current value. Do the same for `window.screenTop`.

screenX
screenY

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `screenX` and `screenY` properties in NN6. Start with the browser window maximized (if you are using Windows). Enter the following property name into the top text box:

```
window.screenY
```

Click the Evaluate button to see the current setting. Unmaximize the window and drag it around the screen. Each time you finish dragging, click the Evaluate button again to see the current value. Do the same for `window.screenY`.

scrollX
scrollY

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `scrollX` and `scrollY` properties in NN6. Enter the following property into the top text box:

```
window.scrollY
```

Now manually scroll the page down so that you can still see the Evaluate button. Click the button to see how far the window has scrolled along the y-axis.

self

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 16-16 uses the same operations as Listing 16-5 but substitutes the `self` property for all `window` object references. The application of this reference is entirely optional, but it can be helpful for reading and debugging scripts if the HTML document is to appear in one frame of a multiframe window — especially if other JavaScript code in this document refers to documents in other frames. The `self` reference helps anyone reading the code know precisely which frame was being addressed.

Listing 16-16: **Using the self Property**

```
<HTML>
<HEAD>
<TITLE>self Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
self.defaultStatus = "Welcome to my Web site."
</SCRIPT>
</HEAD>
<BODY>
<A HREF="http:// www.microsoft.com"
onMouseOver="self.status = 'Visit Microsoft\'s Home page.';return true"
onMouseOut="self.status = '';return true">Microsoft</A><P>
<A HREF="http://home.netscape.com"
onMouseOver="self.status = 'Visit Netscape\'s Home page.';return true"
onMouseOut="self.status = self.defaultStatus;return true">Netscape</A>
</BODY>
</HTML>
```

status

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

In Listing 16-17, the `status` property is set in a handler embedded in the `onMouseOver` attribute of two HTML link tags. Notice that the handler requires a `return true` statement (or any expression that evaluates to `return true`) as the last statement of the handler. This statement is required or the status message will not display, particularly in early browsers.

Listing 16-17: **Links with Custom Statusbar Messages**

```
<HTML>
<HEAD>
<TITLE>window.status Property</TITLE>
</HEAD>
<BODY>
<A HREF="http://www.dannyg.com" onMouseOver="window.status = 'Go to my Home
page. (www.dannyg.com)'; return true">Home</A><P>
<A HREF="http://home.netscape.com" onMouseOver="window.status = 'Visit Netscape
Home page. (home.netscape.com)'; return true">Netscape</A>
</BODY>
</HTML>
```

As a safeguard against platform-specific anomalies that affect the behavior of `onMouseOver` event handlers and the `window.status` property, you should also include an `onMouseOut` event handler for links and client-side image map area objects. Such `onMouseOut` event handlers should set the `status` property to an empty string. This setting ensures that the statusbar message returns to the `defaultStatus` setting when the pointer rolls away from these objects. If you want to write a generalizable function that handles all window status changes, you can do so, but word the `onMouseOver` attribute carefully so that the event handler evaluates to `return true`. Listing 16-18 shows such an alternative.

Listing 16-18: **Handling Status Message Changes**

```
<HTML>
<HEAD>
<TITLE>Generalizable window.status Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function showStatus(msg) {
    window.status = msg
    return true
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="http:// www.dannyg.com " onMouseOver="return showStatus('Go to my Home
page (www.dannyg.com).')" onMouseOut="return showStatus('')">Home</A><P>
<A HREF="http://home.netscape.com" onMouseOver="return showStatus('Visit
Netscape Home page.')" onMouseOut="return showStatus('')">Netscape</A>
</BODY>
</HTML>
```

Notice how the event handlers return the results of the `showStatus()` method to the event handler, allowing the entire handler to evaluate to `return true`.

One final example of setting the statusbar (shown in Listing 16-19) also demonstrates how to create a simple scrolling banner in the statusbar.

Listing 16-19: **Creating a Scrolling Banner**

```
<HTML>
<HEAD>
<TITLE>Message Scroller</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var msg = "Welcome to my world..."
var delay = 150
var timerId
var maxCount = 0
var currCount = 1
```

Continued

Listing 16-19 *(continued)*

```
function scrollMsg() {
    // set the number of times scrolling message is to run
    if (maxCount == 0) {
        maxCount = 3 * msg.length
    }
    window.status = msg
    // keep track of how many characters have scrolled
    currCount++
    // shift first character of msg to end of msg
    msg = msg.substring (1, msg.length) + msg.substring (0, 1)
    // test whether we've reached maximum character count
    if (currCount >= maxCount) {
        timerID = 0        // zero out the timer
        window.status = ""    // clear the status bar
        return            // break out of function
    } else {
        // recursive call to this function
        timerId = setTimeout("scrollMsg()", delay)
   }
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="scrollMsg()">
</BODY>
</HTML>
```

Because the statusbar is being set by a standalone function (rather than by an `onMouseOver` event handler), you do not have to append a `return true` statement to set the `status` property. The `scrollMsg()` function uses more advanced JavaScript concepts, such as the `window.setTimeout()` method (covered later in this chapter) and string methods (covered in Chapter 34 of the *JavaScript Bible*). To speed the pace at which the words scroll across the statusbar, reduce the value of `delay`.

Many Web surfers (myself included) don't care for these scrollers that run forever in the statusbar. Rolling the mouse over links disturbs the banner display. Scrollers can also crash earlier browsers, because the `setTimeout()` method eats application memory in Navigator 2. Use scrolling bars sparingly or design them to run only a few times after the document loads.

Setting the `status` property with `onMouseOver` event handlers has had a checkered career along various implementations in Navigator. A script that sets the statusbar is always in competition against the browser itself, which uses the statusbar to report loading progress. When a "hot" area on a page is at the edge of a frame, many times the `onMouseOut` event fails to fire, thus preventing the statusbar from clearing itself. Be sure to torture test any such implementations before declaring your page ready for public access.

Methods

`alert("message")`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The parameter for the example in Listing 16-20 is a concatenated string. It joins together two fixed strings and the value of the browser's `navigator.appName` property. Loading this document causes the alert dialog box to appear, as shown in several configurations in Figure 2-5. The JavaScript Alert: line cannot be deleted from the dialog box in earlier browsers, nor can the title bar be changed in later browsers.

Listing 16-20: Displaying an Alert Dialog Box

```
<HTML>
<HEAD>
<TITLE>window.alert() Method</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
alert("You are running the " + navigator.appName + " browser.")
</SCRIPT>
</BODY>
</HTML>
```

Figure 2-5: Results of the alert() method in Listing 16-20 in Internet Explorer 5 (top) and Navigator 6 (bottom) for Windows 98

captureEvents(*eventTypeList*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

The page in Listing 16-21 is an exercise in capturing and releasing click events in the `window` object. Whenever the window is capturing click events, the `flash()` function runs. In that function, the event is examined so that only if the Control key is also being held down and the name of the button starts with "button" does the document background color flash red. For all click events (that is, those directed at objects on the page capable of their own `onClick` event handlers), the click is processed with the `routeEvent()` method to make sure the target buttons execute their own `onClick` event handlers.

Listing 16-21: Capturing Click Events in the Window

```
<HTML>
<HEAD>
<TITLE>Window Event Capture</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
// function to run when window captures a click event
function flash(e) {
    if (e.modifiers = Event.CONTROL_MASK &&
    e.target.name.indexOf("button") == 0) {
        document.bgColor = "red"
        setTimeout("document.bgColor = 'white'", 500)
    }
    // let event continue to target
    routeEvent(e)
}
// default setting to capture click events
window.captureEvents(Event.CLICK)
// assign flash() function to click events captured by window
window.onclick = flash
</SCRIPT>
</HEAD>
<BODY BGCOLOR="white">
<FORM NAME="buttons">
<B>Turn window click event capture on or off (Default is "On")</B><P>
<INPUT NAME="captureOn" TYPE="button" VALUE="Capture On"
onClick="window.captureEvents(Event.CLICK)"> 
<INPUT NAME="captureOff" TYPE="button" VALUE="Capture Off"
onClick="window.releaseEvents(Event.CLICK)">
<HR>
<B>Ctrl+Click on a button to see if clicks are being captured by the window
(background color will flash red):</B><P>
<UL>
```

```
<LI><INPUT NAME="button1" TYPE="button" VALUE="Informix" onClick="alert('You
clicked on Informix.')">
<LI><INPUT NAME="button2" TYPE="button" VALUE="Oracle" onClick="alert('You
clicked on Oracle.')">
<LI><INPUT NAME="button3" TYPE="button" VALUE="Sybase" onClick="alert('You
clicked on Sybase.')">
</UL>
</FORM>
</BODY>
</HTML>
```

When you try this page, also turn off window event capture. Now only the buttons' `onClick` event handlers execute, and the page does not flash red.

clearInterval(*intervalIDnumber*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

See Listings 16-36 and 16-37 for an example of how `setInterval()` and `clearInterval()` are used together on a page.

clearTimeout(*timeoutIDnumber*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The page in Listing 16-22 features one text field and two buttons (Figure 2-6). One button starts a countdown timer coded to last one minute (easily modifiable for other durations); the other button interrupts the timer at any time while it is running. When the minute is up, an alert dialog box lets you know.

Listing 16-22: **A Countdown Timer**

```
<HTML>
<HEAD>
<TITLE>Count Down Timer</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var running = false
```

Continued

Listing 16-22 *(continued)*

```
var endTime = null
var timerID = null

function startTimer() {
    running = true
    now = new Date()
    now = now.getTime()
    // change last multiple for the number of minutes
    endTime = now + (1000 * 60 * 1)
    showCountDown()
}

function showCountDown() {
    var now = new Date()
    now = now.getTime()
    if (endTime - now <= 0) {
        stopTimer()
        alert("Time is up.  Put down your pencils.")
    } else {
        var delta = new Date(endTime - now)
        var theMin = delta.getMinutes()
        var theSec = delta.getSeconds()
        var theTime = theMin
        theTime += ((theSec < 10) ? ":0" : ":") + theSec
        document.forms[0].timerDisplay.value = theTime
        if (running) {
            timerID = setTimeout("showCountDown()",1000)
        }
    }
}

function stopTimer() {
    clearTimeout(timerID)
    running = false
    document.forms[0].timerDisplay.value = "0:00"
}
//-->
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button" NAME="startTime" VALUE="Start 1 min. Timer"
onClick="startTimer()">
<INPUT TYPE="button" NAME="clearTime" VALUE="Clear Timer"
onClick="stopTimer()"><P>
<INPUT TYPE="text" NAME="timerDisplay" VALUE="">
</FORM>
</BODY>
</HTML>
```

Notice that the script establishes three variables with global scope in the window: `running`, `endTime`, and `timerID`. These values are needed inside multiple functions, so they are initialized outside of the functions.

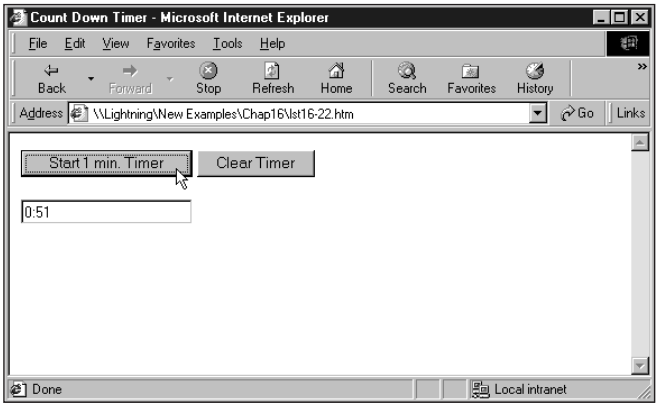


Figure 2-6: The countdown timer page as it displays the time remaining

In the `startTimer()` function, you switch the running flag on, meaning that the timer should be going. Using some date functions (see Chapter 36 of the *JavaScript Bible*), you extract the current time in milliseconds and add the number of milliseconds for the next minute (the extra multiplication by one is the place where you can change the amount to the desired number of minutes). With the end time stored in a global variable, the function now calls another function that compares the current and end times and displays the difference in the text field.

Early in the `showCountDown()` function, check to see if the timer has wound down. If so, you stop the timer and alert the user. Otherwise, the function continues to calculate the difference between the two times and formats the time in mm:ss format. As long as the `running` flag is set to `true`, the function sets the one-second timeout timer before repeating itself. To stop the timer before it has run out (in the `stopTimer()` function), the most important step is to cancel the timeout running inside the browser. The `clearTimeout()` method uses the global `timerID` value to do that. Then the function turns off the running switch and zeros out the display.

When you run the timer, you may occasionally notice that the time skips a second. It's not cheating. It just takes slightly more than one second to wait for the timeout and then finish the calculations for the next second's display. What you're seeing is the display catching up with the real time left.

close()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listing 16-4 (for the `window.closed` property), which provides an elaborate, cross-platform, bug-accommodating example of applying the `window.close()` method across multiple windows.

`confirm("message")`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The example in Listing 16-23 shows the user interface part of how you can use a confirm dialog box to query a user before clearing a table full of user-entered data. The line in the title bar, as shown in Figure 2-7, or the "JavaScript Confirm" legend in earlier browser versions, cannot be removed from the dialog box.

Listing 16-23: **The Confirm Dialog Box**

```
<HTML>
<HEAD>
<TITLE>window.confirm() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function clearTable() {
    if (confirm("Are you sure you want to empty the table?")) {
        alert("Emptying the table...") // for demo purposes
        //statements that actually empty the fields
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<!-- other statements that display and populate a large table -->
<INPUT TYPE="button" NAME="clear" VALUE="Reset Table" onClick="clearTable()">
</FORM>
</BODY>
</HTML>
```

Figure 2-7: A JavaScript confirm dialog box (IE5/Windows format)

createPopup()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

See Listing 16-49 later in this chapter for an example of the `createPopup()` method.

disableExternalCapture()
enableExternalCapture()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

As this was a little-used feature of NN4 even while the browser enjoyed a substantial installed base, it becomes less important as that browser version recedes into history. You can find an example of this feature at the Support Center for this book (`http://www.dannyg.com/update.html`) or on pp.213–214 of the *JavaScript Bible,* 3rd edition.

execScript("*exprList*"[, *language*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `execScript()` method. The Evaluator has predeclared global variables for the lowercase letters `a` through `z`. Enter each of the following statements into the top text box and observe the results for each.

```
a
```

When first loaded, the variable is declared but assigned no value, so it is undefined.

```
window.execScript("a = 5")
```

The method returns no value, so the mechanism inside The Evaluator says that the statement is undefined.

```
a
```

The variable is now 5.

```
window.execScript("b = a * 50")
b
```

The `b` global variable has a value of 250. Continue exploring with additional script statements. Use semicolons to separate multiple statements within the string parameter.

find(["*searchString*" [, *matchCaseBoolean*, *searchUpBoolean*]])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

A simple call to the `window.find()` method looks as follows:

```
var success = window.find("contract")
```

If you want the search to be case-sensitive, add at least one of the two optional parameters:

```
success = wind.find(matchString,caseSensitive,backward)
```

Because this method works only in NN4, refer to discussions of the `TextRange` and `Range` objects in Chapter 19 of the *JavaScript Bible* for more modern implementations of body text searching.

GetAttention()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) in NN6 to set a timer that gives you enough time to switch to another application and wait for the attention signal to fire. Enter the following statement into the top text box, click the Evaluate button, and then quickly switch to another program:

```
setTimeout("GetAttention()", 5000)
```

After a total of five seconds, the attention signal fires.

moveBy(*deltaX*,*deltaY*)
moveTo(*x*,*y*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Several examples of using the `window.moveTo()` and `window.moveBy()` methods are shown in Listing 16-24. The page presents four buttons, each of which performs a different kind of browser window movement.

Listing 16-24: **Window Boogie**

```
<HTML>
<HEAD>
<TITLE>Window Gymnastics</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
var isNav4 = ((navigator.appName == "Netscape") &&
(parseInt(navigator.appVersion) >= 4))
// wait in onLoad for page to load and settle in IE
function init() {
    // fill missing IE properties
    if (!window.outerWidth) {
        window.outerWidth = document.body.clientWidth
        window.outerHeight = document.body.clientHeight + 30
    }
    // fill missing IE4 properties
    if (!screen.availWidth) {
        screen.availWidth = 640
        screen.availHeight = 480
    }
}
// function to run when window captures a click event
function moveOffScreen() {
    // branch for NN security
    if (isNav4) {
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite")
    }
    var maxX = screen.width
    var maxY = screen.height
    window.moveTo(maxX+1, maxY+1)
    setTimeout("window.moveTo(0,0)",500)
    if (isNav4) {
netscape.security.PrivilegeManager.disablePrivilege("UniversalBrowserWrite")
    }
```

Continued

Listing 16-24 *(continued)*

```
}
// moves window in a circular motion
function revolve() {
    var winX = (screen.availWidth - window.outerWidth) / 2
    var winY = 50
    window.resizeTo(400,300)
    window.moveTo(winX, winY)

    for (var i = 1; i < 36; i++) {
        winX += Math.cos(i * (Math.PI/18)) * 5
        winY += Math.sin(i * (Math.PI/18)) * 5
        window.moveTo(winX, winY)
    }
}
// moves window in a horizontal zig-zag pattern
function zigzag() {
    window.resizeTo(400,300)
    window.moveTo(0,80)
    var incrementX = 2
    var incrementY = 2
    var floor = screen.availHeight - window.outerHeight
    var rightEdge = screen.availWidth - window.outerWidth
    for (var i = 0; i < rightEdge; i += 2) {
        window.moveBy(incrementX, incrementY)
        if (i%60 == 0) {
            incrementY = -incrementY
        }
    }
}
// resizes window to occupy all available screen real estate
function maximize() {
    window.moveTo(0,0)
    window.resizeTo(screen.availWidth, screen.availHeight)
}
</SCRIPT>
</HEAD>
<BODY onLoad="init()">
<FORM NAME="buttons">
<B>Window Gymnastics</B><P>
<UL>
<LI><INPUT NAME="offscreen" TYPE="button" VALUE="Disappear a Second"
onClick="moveOffScreen()">
<LI><INPUT NAME="circles" TYPE="button" VALUE="Circular Motion"
onClick="revolve()">
<LI><INPUT NAME="bouncer" TYPE="button" VALUE="Zig Zag" onClick="zigzag()">
<LI><INPUT NAME="expander" TYPE="button" VALUE="Maximize" onClick="maximize()">
</UL>
</FORM>
</BODY>
</HTML>
```

To run successfully in NN, the first button requires that you have codebase principals turned on (see Chapter 46 of the *JavaScript Bible*) to take advantage of what would normally be a signed script. The `moveOffScreen()` function momentarily moves the window entirely out of view. Notice how the script determines the size of the screen before deciding where to move the window. After the journey off screen, the window comes back into view at the upper-left corner of the screen.

If using the Web sometimes seems like going around in circles, then the second function, `revolve()`, should feel just right. After reducing the size of the window and positioning it near the top center of the screen, the script uses a bit of math to position the window along 36 places around a perfect circle (at 10-degree increments). This is an example of how to control a window's position dynamically based on math calculations. IE complicates the job a bit by not providing properties that reveal the outside dimensions of the browser window.

To demonstrate the `moveBy()` method, the third function, `zigzag()`, uses a `for` loop to increment the coordinate points to make the window travel in a saw tooth pattern across the screen. The x coordinate continues to increment linearly until the window is at the edge of the screen (also calculated on the fly to accommodate any size monitor). The y coordinate must increase and decrease as that parameter changes direction at various times across the screen.

In the fourth function, you see some practical code (finally) that demonstrates how best to simulate maximizing the browser window to fill the entire available screen space on the visitor's monitor.

navigate("*URL*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility					✓	✓	✓	✓	✓

Example

Supply any valid URL as the parameter to the method, as in

```
window.navigate("http://www.dannyg.com")
```

open("*URL*", "*windowName*" [, "*windowFeatures*"][,*replaceFlag*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The page rendered by Listing 16-26 displays a single button that generates a new window of a specific size that has only the statusbar turned on. The script here

shows all the elements necessary to create a new window that has all the right stuff on most platforms. The new window object reference is assigned to a global variable, `newWindow`. Before a new window is generated, the script looks to see if the window has never been generated before (in which case `newWindow` would be `null`) or, for newer browsers, the window is closed. If either condition is true, the window is created with the `open()` method. Otherwise, the existing window is brought forward with the `focus()` method (NN3+ and IE4+).

As a safeguard against older browsers, the script manually adds an `opener` property to the new window if one is not already assigned by the `open()` method. The current `window` object reference is assigned to that property.

Due to the timing problem that afflicts all IE generations, the HTML assembly and writing to the new window is separated into its own function that is invoked after a 50 millisecond delay (NN goes along for the ride, but it could accommodate the assembly and writing without the delay). To build the string that is eventually written to the document, I use the += (add-by-value) operator, which appends the string on the right side of the operator to the string stored in the variable on the left side. In this example, the new window is handed an `<H1>`-level line of text to display.

Listing 16-26: **Creating a New Window**

```
<HTML>
<HEAD>
<TITLE>New Window</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var newWindow
function makeNewWindow() {
    if (!newWindow || newWindow.closed) {
        newWindow = window.open("","","status,height=200,width=300")
        if (!newWindow.opener) {
        newWindow.opener = window
        }
        // force small delay for IE to catch up
        setTimeout("writeToWindow()", 50)
    } else {
        // window's already open; bring to front
        newWindow.focus()
    }
}
function writeToWindow() {
    // assemble content for new window
    var newContent = "<HTML><HEAD><TITLE>One Sub Window</TITLE></HEAD>"
    newContent += "<BODY><H1>This window is brand new.</H1>"
    newContent += "</BODY></HTML>"
    // write HTML to new window document
    newWindow.document.write(newContent)
    newWindow.document.close() // close layout stream
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
```

```
<INPUT TYPE="button" NAME="newOne" VALUE="Create New Window"
 onClick="makeNewWindow()">
</FORM>
</BODY>
</HTML>
```

If you need to create a new window for the lowest common denominator of scriptable browser, you will have to omit the `focus()` method and the `window.closed` property from the script (as well as add the NN2 bug workaround described earlier). Or you may prefer to forego a subwindow for all browsers below a certain level. See Listing 16-3 (in the `window.closed` property discussion) for other ideas about cross-browser authoring for subwindows.

print()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓				✓	✓

Example

Listing 16-27 is a frameset that loads Listing 16-28 into the top frame and a copy of the Bill of Rights into the bottom frame.

Listing 16-27: **Print Frameset**

```
<HTML>
<HEAD>
<TITLE>window.print() method</TITLE>
</HEAD>
<FRAMESET ROWS="25%,75%">
    <FRAME NAME="controls" SRC="lst16-28.htm">
    <FRAME NAME="display" SRC="bofright.htm">
</FRAMESET>
</HTML>
```

Two buttons in the top control panel (Listing 16-28) let you print the whole frameset (in those browsers and OSs that support it) or just the lower frame. To print the entire frameset, the reference includes the parent window; to print the lower frame, the reference is directed at the `parent.display` frame.

Listing 16-28: **Printing Control**

```
<HTML>
<HEAD>
<TITLE>Print()</TITLE>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" NAME="printWhole" VALUE="Print Entire Frameset"
onClick="parent.print()"><P>
<INPUT TYPE="button" NAME="printFrame" VALUE="Print Bottom Frame Only"
onClick="parent.display.print()"><P>
</FORM>
</BODY>
</HTML>
```

If you don't like some facet of the printed output, blame the browser's print engine, and not JavaScript. The `print()` method merely invokes the browser's regular printing routines. Pages whose content is generated entirely by JavaScript print only in NN3+ and IE4+.

`prompt("message", "defaultReply")`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The function that receives values from the prompt dialog box in Listing 16-29 (see the dialog box in Figure 2-8) does some data-entry validation (but certainly not enough for a commercial site). The function first checks to make sure that the returned value is neither `null` (Cancel) nor an empty string (the user clicked OK without entering any values). See Chapter 43 of the *JavaScript Bible* for more about data-entry validation.

Listing 16-29: **The Prompt Dialog Box**

```
<HTML>
<HEAD>
<TITLE>window.prompt() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function populateTable() {
    var howMany = prompt("Fill in table for how many factors?","")
    if (howMany != null && howMany != "") {
```

```
        alert("Filling the table for " + howMany) // for demo
        //statements that validate the entry and
        //actually populate the fields of the table
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<!-- other statements that display and populate a large table -->
<INPUT TYPE="button" NAME="fill" VALUE="Fill Table..."
onClick="populateTable()">
</FORM>
</BODY>
</HTML>
```

Figure 2-8: The prompt dialog box displayed from Listing 16-29 (Windows format)

Notice one important user interface element in Listing 16-29. Because clicking the button leads to a dialog box that requires more information from the user, the button's label ends in an ellipsis (or, rather, three periods acting as an ellipsis character). The ellipsis is a common courtesy to let users know that a user interface element leads to a dialog box of some sort. As in similar situations in Windows and Macintosh programs, the user should be able to cancel out of that dialog box and return to the same screen state that existed before the button was clicked.

resizeBy(*deltaX*,*deltaY*)
resizeTo(*outerwidth*,*outerheight*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

You can experiment with the resize methods with the page in Listing 16-30. Two parts of a form let you enter values for each method. The one for `window.resize()` also lets you enter a number of repetitions to better see the impact of the values. Enter zero and negative values to see how those affect the method. Also test the limits of different browsers.

Listing 16-30: **Window Resize Methods**

```
<HTML>
<HEAD>
<TITLE>Window Resize Methods</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doResizeBy(form) {
    var x = parseInt(form.resizeByX.value)
    var y = parseInt(form.resizeByY.value)
    var count = parseInt(form.count.value)
    for (var i = 0; i < count; i++) {
        window.resizeBy(x, y)
    }
}
function doResizeTo(form) {
    var x = parseInt(form.resizeToX.value)
    var y = parseInt(form.resizeToY.value)
    window.resizeTo(x, y)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<B>Enter the x and y increment, plus how many times the window should be resized
by these increments:</B><BR>
Horiz:<INPUT TYPE="text" NAME="resizeByX" SIZE=4>
Vert:<INPUT TYPE="text" NAME="resizeByY" SIZE=4>
How Many:<INPUT TYPE="text" NAME="count" SIZE=4>
<INPUT TYPE="button" NAME="ResizeBy" VALUE="Show resizeBy()"
onClick="doResizeBy(this.form)">
<HR>
<B>Enter the desired width and height of the current window:</B><BR>
Width:<INPUT TYPE="text" NAME="resizeToX" SIZE=4>
Height:<INPUT TYPE="text" NAME="resizeToY" SIZE=4>
<INPUT TYPE="button" NAME="ResizeTo" VALUE="Show resizeTo()"
onClick="doResizeTo(this.form)">
</FORM>
</BODY>
</HTML>
```

routeEvent(*event*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

The `window.routeEvent()` method is used in the example for `window.captureEvents()`, Listing 16-21.

scroll(*horizontalCoord, verticalCoord*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

To demonstrate the `scroll()` method, Listing 16-31 defines a frameset with a document in the top frame (Listing 16-32) and a control panel in the bottom frame (Listing 16-33). A series of buttons and text fields in the control panel frame directs the scrolling of the document. I've selected an arbitrary, large GIF image to use in the example. To see results of some horizontal scrolling values, you may need to shrink the width of the browser window until a horizontal scrollbar appears in the top frame. Figure 2-9 shows the results in a shrunken window with modest horizontal and vertical scroll values entered into the bottom text boxes. If you substitute `scrollTo()` for the `scroll()` methods in Listing 16-33, the results will be the same, but you will need version browsers at a minimum to run it.

Listing 16-31: **A Frameset for the scroll() Demonstration**

```
<HTML>
<HEAD>
<TITLE>window.scroll() Method</TITLE>
</HEAD>

<FRAMESET ROWS="50%,50%">
    <FRAME SRC="lst16-32.htm" NAME="display">
    <FRAME SRC="lst16-33.htm" NAME="control">
</FRAMESET>
</HTML>
```

Listing 16-32: **The Image to Be Scrolled**

```
<HTML>
<HEAD>
<TITLE>Arch</TITLE>
</HEAD>
```

Continued

Listing 16-32 *(continued)*

```
<BODY>
<H1>A Picture is Worth...</H1>
<HR>
<CENTER>
<TABLE BORDER=3>
<CAPTION ALIGN=bottom>A Splendid Arch</CAPTION>
<TD>
<IMG SRC="arch.gif">
</TD></TABLE></CENTER>
</BODY>
</HTML>
```

Listing 16-33: **Controls to Adjust Scrolling of the Upper Frame**

```
<HTML>
<HEAD>
<TITLE>Scroll Controller</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
function scroll(x,y) {
    parent.frames[0].scroll(x,y)
}
function customScroll(form) {
    parent.frames[0].scroll(parseInt(form.x.value),parseInt(form.y.value))
}
</SCRIPT>
</HEAD>
<BODY>
<H2>Scroll Controller</H2>
<HR>
<FORM NAME="fixed">
Click on a scroll coordinate for the upper frame:<P>
<INPUT TYPE="button" VALUE="0,0" onClick="scroll(0,0)">
<INPUT TYPE="button" VALUE="0,100" onClick="scroll(0,100)">
<INPUT TYPE="button" VALUE="100,0" onClick="scroll(100,0)">
<P>
<INPUT TYPE="button" VALUE="-100,100" onClick="scroll(-100,100)">
<INPUT TYPE="button" VALUE="20,200" onClick="scroll(20,200)">
<INPUT TYPE="button" VALUE="1000,3000" onClick="scroll(1000,3000)">
</FORM>
<HR>
<FORM NAME="custom">
Enter a Horizontal
<INPUT TYPE="text" NAME="x" VALUE="0" SIZE=4>
and Vertical
<INPUT TYPE="text" NAME="y" VALUE="0" SIZE=4>
value.  Then
```

```
<INPUT TYPE="button" VALUE="click to scroll" onClick="customScroll(this.form)">
</FORM>
</BODY>
</HTML>
```

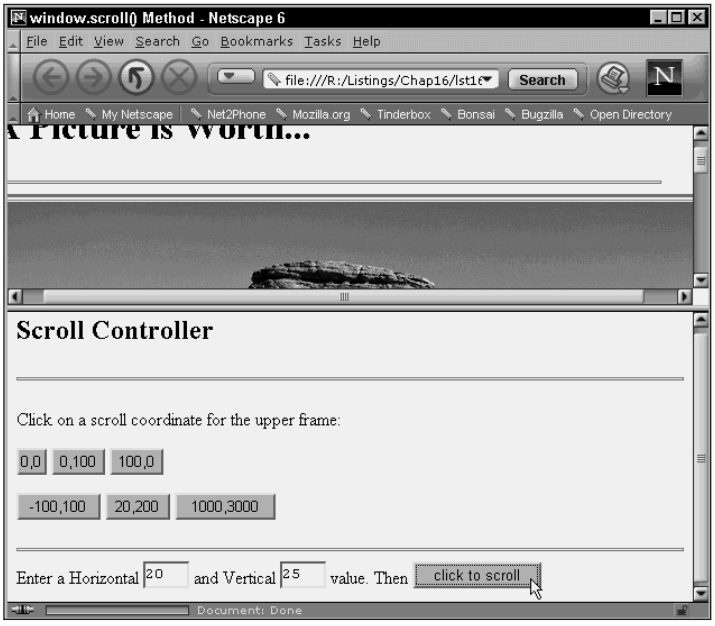


Figure 2-9: Scripts control the scrolling of the top frame

Notice that in the `customScroll()` function, JavaScript must convert the string values from the two text boxes to integers (with the `parseInt()` method) for the `scroll()` method to accept them. Nonnumeric data can produce very odd results. Also be aware that although this example shows how to adjust the scroll values in another frame, you can set such values in the same frame or window as the script, as well as in subwindows, provided that you use the correct object references to the window.

scrollBy(*deltaX*,*deltaY*)
scrollTo(*x*,*y*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

To work with the `scrollTo()` method, you can use Listings 16-31 through 16-33 (the `window.scroll()` method) but substitute `window.scrollTo()` for `window.scroll()`. The results should be the same. For `scrollBy()`, the example starts with the frameset in Listing 16-34. It loads the same content document as the `window.scroll()` example (Listing 16-32), but the control panel (Listing 16-35) provides input to experiment with the `scrollBy()` method.

Listing 16-34: **Frameset for ScrollBy Controller**

```
<HTML>
<HEAD>
<TITLE>window.scrollBy() Method</TITLE>
</HEAD>

<FRAMESET ROWS="50%,50%">
    <FRAME SRC="lst16-32.htm" NAME="display">
    <FRAME SRC="lst16-35.htm" NAME="control">
</FRAMESET>
</HTML>
```

Notice in Listing 16-35 that all references to window properties and methods are directed to the `display` frame. String values retrieved from text fields are converted to number with the `parseInt()` global function.

Listing 16-35: **ScrollBy Controller**

```
<HTML>
<HEAD>
<TITLE>ScrollBy Controller</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function page(direction) {
    var pixFrame = parent.display
    var deltaY = (pixFrame.innerHeight) ? pixFrame.innerHeight :
        pixFrame.document.body.scrollHeight
    if (direction == "up") {
        deltaY = -deltaY
    }
    parent.display.scrollBy(0, deltaY)
}
function customScroll(form) {
    parent.display.scrollBy(parseInt(form.x.value), parseInt(form.y.value))
}
</SCRIPT>
</HEAD>
<BODY>
<B>ScrollBy Controller</B>
<FORM NAME="custom">
Enter an Horizontal increment
```

```
<INPUT TYPE="text" NAME="x" VALUE="0" SIZE=4">
and Vertical
<INPUT TYPE="text" NAME="y" VALUE="0" SIZE=4">
value.<BR>Then
<INPUT TYPE="button" VALUE="click to scrollBy()"
onClick="customScroll(this.form)">
<HR>
<INPUT TYPE="button" VALUE="PageDown" onClick="page('down')">
<INPUT TYPE="button" VALUE="PageUp" onClick="page('up')">

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

setCursor("*cursorType*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) in NN6 to experiment with setting the cursor. After clicking the top text box in preparation for typing, roll the cursor to a location atop an empty spot on the page. Then enter the following statements one at a time into the top text box and press Enter/Return:

```
setCursor("wait")
setCursor("spinning"
setCursor("move")
```

After evaluating each statement, roll the cursor around the page, and notice where the cursor reverts to its normal appearance.

setInterval("*expr*", *msecDelay* [, *language*])
setInterval(*funcRef*, *msecDelay* [, *funcarg1*, ..., *funcargn*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

The demonstration of the `setInterval()` method entails a two-framed environment. The framesetting document is shown in Listing 16-36.

Listing 16-36: setInterval() Demonstration Frameset

```
<HTML>
<HEAD>
<TITLE>setInterval() Method</TITLE>
</HEAD>

<FRAMESET ROWS="50%,50%">
    <FRAME SRC="lst16-37.htm" NAME="control">
    <FRAME SRC="bofright.htm" NAME="display">
</FRAMESET>
</HTML>
```

In the top frame is a control panel with several buttons that control the automatic scrolling of the Bill of Rights text document in the bottom frame. Listing 16-37 shows the control panel document. Many functions here control the interval, scrolling jump size, and direction, and they demonstrate several aspects of applying `setInterval()`.

Notice that in the beginning the script establishes a number of global variables. Three of them are parameters that control the scrolling; the last one is for the ID value returned by the `setInterval()` method. The script needs that value to be a global value so that a separate function can halt the scrolling with the `clearInterval()` method.

All scrolling is performed by the `autoScroll()` function. For the sake of simplicity, all controlling parameters are global variables. In this application, placement of those values in global variables helps the page restart autoscrolling with the same parameters as it had when it last ran.

Listing 16-37: setInterval() Control Panel

```
<HTML>
<HEAD>
<TITLE>ScrollBy Controller</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
var scrollSpeed = 500
var scrollJump = 1
var scrollDirection = "down"
var intervalID

function autoScroll() {
    if (scrollDirection == "down") {
        scrollJump = Math.abs(scrollJump)
    } else if (scrollDirection == "up" && scrollJump > 0) {
        scrollJump = -scrollJump
    }
    parent.display.scrollBy(0, scrollJump)
    if (parent.display.pageYOffset <= 0) {
```

```
        clearInterval(intervalID)
    }
}

function reduceInterval() {
    stopScroll()
    scrollSpeed -= 200
    startScroll()
}
function increaseInterval() {
    stopScroll()
    scrollSpeed += 200
    startScroll()
}
function reduceJump() {
    scrollJump -= 2
}
function increaseJump() {
    scrollJump += 2
}
function swapDirection() {
    scrollDirection = (scrollDirection == "down") ? "up" : "down"
}
function startScroll() {
    parent.display.scrollBy(0, scrollJump)
    if (intervalID) {
        clearInterval(intervalID)
    }
    intervalID = setInterval("autoScroll()",scrollSpeed)
}
function stopScroll() {
    clearInterval(intervalID)
}
</SCRIPT>
</HEAD>
<BODY onLoad="startScroll()">
<B>AutoScroll by setInterval() Controller</B>
<FORM NAME="custom">
<INPUT TYPE="button" VALUE="Start Scrolling" onClick="startScroll()">
<INPUT TYPE="button" VALUE="Stop Scrolling" onClick="stopScroll()"><P>
<INPUT TYPE="button" VALUE="Shorter Time Interval" onClick="reduceInterval()">
<INPUT TYPE="button" VALUE="Longer Time Interval"
onClick="increaseInterval()"><P>
<INPUT TYPE="button" VALUE="Bigger Scroll Jumps" onClick="increaseJump()">
<INPUT TYPE="button" VALUE="Smaller Scroll Jumps" onClick="reduceJump()"><P>
<INPUT TYPE="button" VALUE="Change Direction" onClick="swapDirection()">

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

The `setInterval()` method is invoked inside the `startScroll()` function. This function initially "burps" the page by one `scrollJump` interval so that the test in `autoScroll()` for the page being scrolled all the way to the top doesn't halt a page from scrolling before it gets started. Notice, too, that the function checks for the existence of an interval ID. If one is there, it is cleared before the new one is set. This is crucial within the design of the example page, because repeated clicking of the Start Scrolling button triggers multiple interval timers inside the browser. Only the most recent one's ID would be stored in `intervalID`, allowing no way to clear the older ones. But this little side trip makes sure that only one interval timer is running. One of the global variables, `scrollSpeed`, is used to fill the delay parameter for `setInterval()`. To change this value on the fly, the script must stop the current interval process, change the `scrollSpeed` value, and start a new process. The intensely repetitive nature of this application is nicely handled by the `setInterval()` method.

setTimeout("*expr*", *msecDelay* [, *language*])
setTimeout(*functionRef*, *msecDelay* [, *funcarg1*, ..., *funcargn*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

When you load the HTML page in Listing 16-38, it triggers the `updateTime()` function, which displays the time (in hh:mm am/pm format) in the statusbar. Instead of showing the seconds incrementing one by one (which may be distracting to someone trying to read the page), this function alternates the last character of the display between an asterisk and nothing, like a visual "heartbeat."

Listing 16-38: **Display the Current Time**

```
<HTML>
<HEAD>
<TITLE>Status Bar Clock</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var flasher = false
// calculate current time, determine flasher state,
// and insert time into status bar every second
function updateTime() {
    var now = new Date()
    var theHour = now.getHours()
    var theMin = now.getMinutes()
    var theTime = "" + ((theHour > 12) ? theHour - 12 : theHour)
    theTime += ((theMin < 10) ? ":0" : ":") + theMin
```

```
    theTime  += (theHour >= 12) ? " pm" : " am"
    theTime += ((flasher) ? " " : "*")
    flasher = !flasher
    window.status = theTime
    // recursively call this function every second to keep timer going
    timerID = setTimeout("updateTime()",1000)
}
//-->
</SCRIPT>
</HEAD>

<BODY onLoad="updateTime()">
</BODY>
</HTML>
```

In this function, the `setTimeout()` method works in the following way: Once the current time (including the flasher status) appears in the statusbar, the function waits approximately one second (1,000 milliseconds) before calling the same function again. You don't have to clear the `timerID` value in this application because JavaScript does it for you every time the 1,000 milliseconds elapse.

A logical question to ask is whether this application should be using `setInterval()` instead of `setTimeout()`. This is a case in which either one does the job. To use `setInterval()` here would require that the interval process start outside of the `updateTime()` function, because you need only one process running that repeatedly calls `updateTime()`. It would be a cleaner implementation in that regard, instead of the tons of timeout processes spawned by Listing 16-38. On the other hand, the application would not run in any browsers before NN4 or IE4, as Listing 16-38 does.

To demonstrate passing parameters, you can modify the `updateTime()` function to add the number of times it gets invoked to the display in the statusbar. For that to work, the function must have a parameter variable so that it can catch a new value each time it is invoked by `setTimeout()`'s expression. For all browsers, the function would be modified as follows (unchanged lines are represented by the ellipsis):

```
function updateTime(i) {
    ...
    window.status = theTime + "  (" + i + ")"
    // pass updated counter value with next call to this function
    timerID = setTimeout("updateTime(" + i+1 + ")",1000)
}
```

If you were running this exclusively in NN4+, you could use its more convenient way of passing parameters to the function:

```
timerID = setTimeout(updateTime,1000, i+1)
```

In either case, the `onLoad` event handler would also have to be modified to get the ball rolling with an initial parameter:

```
onLoad = "updateTime(0)"
```

One warning about `setTimeout()` functions that dive into themselves as frequently as this one does: Each call eats up a bit more memory for the browser application in Navigator 2. If you let this clock run for a while, some browsers may encounter memory difficulties, depending on which operating system they're using. But considering the amount of time the typical user spends on Web pages (even if only 10 or 15 minutes), the function shouldn't present a problem. And any reloading invoked by the user (such as by resizing the window in Navigator 2) frees up memory once again.

```
showModalDialog("URL"[, arguments]
[, features])
showModelessDialog("URL"[, arguments]
[, features])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							(✓)	✓	✓

Example

To demonstrate the two styles of dialog boxes, I have implemented the same functionality (setting some session visual preferences) for both modal and modeless dialog boxes. This tactic shows you how to pass data back and forth between the main page and both styles of dialog box windows.

The first example demonstrates how to use a modal dialog box. In the process, data is passed into the dialog box window and values are returned. Listing 16-39 is the HTML and scripting for the main page. A button's `onClick` event handler invokes a function that opens the modal dialog box. The dialog box's document (Listing 16-40) contains several form elements for entering a user name and selecting a few color styles for the main page. Data from the dialog is fashioned into an array to be sent back to the main window. That array is initially assigned to a local variable, `prefs`, as the dialog box closes. If the user cancels the dialog box, the returned value is an empty string, so nothing more in `getPrefsData()` executes. But when the user clicks OK, the array comes back. Each of the array items is read and assigned to its respective form value or style property. These values are also preserved in the global `currPrefs` array. This allows the settings to be sent to the modal dialog box (as the second parameter to `showModalDialog()`) the next time the dialog box is opened.

Listing 16-39: **Main Page for showModalDialog()**

```
<HTML>
<HEAD>
<TITLE>window.setModalDialog() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var currPrefs = new Array()
```

```
function getPrefsData() {
    var prefs = showModalDialog("lst16-40.htm", currPrefs,
        "dialogWidth:400px; dialogHeight:300px")
    if (prefs) {
        if (prefs["name"]) {
            document.all.firstName.innerText = prefs["name"]
            currPrefs["name"] = prefs["name"]
        }
        if (prefs["bgColor"]) {
            document.body.style.backgroundColor = prefs["bgColor"]
            currPrefs["bgColor"] = prefs["bgColor"]
        }
        if (prefs["textColor"]) {
            document.body.style.color = prefs["textColor"]
            currPrefs["textColor"] = prefs["textColor"]
        }
        if (prefs["h1Size"]) {
            document.all.welcomeHeader.style.fontSize = prefs["h1Size"]
            currPrefs["h1Size"] = prefs["h1Size"]
        }
    }
}
function init() {
    document.all.firstName.innerText = "friend"
}
</SCRIPT>

</HEAD>
<BODY BGCOLOR="#eeeeee" STYLE="margin:20px" onLoad="init()">
<H1>window.setModalDialog() Method</H1>
<HR>
<H2 ID="welcomeHeader">Welcome, <SPAN ID="firstName"> </SPAN>!</H2>
<HR>
<P>Use this button to set style preferences for this page:
<BUTTON ID="prefsButton" onClick="getPrefsData()">
Preferences
</BUTTON>
</BODY>
</HTML>
```

The dialog box's document, shown in Listing 16-40, is responsible for reading the incoming data (and setting the form elements accordingly) and assembling form data for return to the main window's script. Notice when you load the example that the TITLE element of the dialog box's document appears in the dialog box window's title bar.

When the page loads into the dialog box window, the `init()` function examines the `window.dialogArguments` property. If it has any data, the data is used to preset the form elements to mirror the current settings of the main page. A utility function, `setSelected()`, pre-selects the option of a SELECT element to match the current settings.

Buttons at the bottom of the page are explicitly positioned to be at the lower-right corner of the window. Each button invokes a function to do what is needed to close the dialog box. In the case of the OK button, the `handleOK()` function sets the `window.returnValue` property to the data that come back from the `getFormData()` function. This latter function reads the form element values and packages them in an array using the form elements' names as array indices. This helps keep everything straight back in the main window's script, which uses the index names, and is therefore not dependent upon the precise sequence of the form elements in the dialog box window.

Listing 16-40: **Document for the Modal Dialog**

```
<HTML>
<HEAD>
<TITLE>User Preferences</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// Close the dialog
function closeme() {
    window.close()
}

// Handle click of OK button
function handleOK() {
    window.returnValue = getFormData()
    closeme()
}

// Handle click of Cancel button
function handleCancel() {
    window.returnValue = ""
    closeme()
}
// Generic function converts form element name-value pairs
// into an array
function getFormData() {
    var form = document.prefs
    var returnedData = new Array()
    // Harvest values for each type of form element
    for (var i = 0; i < form.elements.length; i++) {
        if (form.elements[i].type == "text") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else if (form.elements[i].type.indexOf("select") != -1) {
            returnedData[form.elements[i].name] =
            form.elements[i].options[form.elements[i].selectedIndex].value
        } else if (form.elements[i].type == "radio") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else if (form.elements[i].type == "checkbox") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else continue
    }
    return returnedData
}
```

```
// Initialize by setting form elements from passed data
function init() {
    if (window.dialogArguments) {
        var args = window.dialogArguments
        var form = document.prefs
        if (args["name"]) {
            form.name.value = args["name"]
        }
        if (args["bgColor"]) {
            setSelected(form.bgColor, args["bgColor"])
        }
        if (args["textColor"]) {
            setSelected(form.textColor, args["textColor"])
        }
        if (args["h1Size"]) {
            setSelected(form.h1Size, args["h1Size"])
        }
    }
}
// Utility function to set a SELECT element to one value
function setSelected(select, value) {
    for (var i = 0; i < select.options.length; i++) {
        if (select.options[i].value == value) {
            select.selectedIndex = i
            break
        }
    }
    return
}
// Utility function to accept a press of the
// Enter key in the text field as a click of OK
function checkEnter() {
    if (window.event.keyCode == 13) {
        handleOK()
    }
}
</SCRIPT>
</HEAD>

<BODY BGCOLOR="#eeeeee" onLoad="init()">
<H2>Web Site Preferences</H2>
<HR>
<TABLE BORDER=0 CELLSPACING=2>
<FORM NAME="prefs" onSubmit="return false">
<TR>
<TD>Enter your first name:<INPUT NAME="name" TYPE="text" VALUE="" SIZE=20
onKeyDown="checkEnter()">
</TR>

<TR>
<TD>Select a background color:
<SELECT NAME="bgColor">
    <OPTION VALUE="beige">Beige
    <OPTION VALUE="antiquewhite">Antique White
```

Continued

Listing 16-40 *(continued)*

```
    <OPTION VALUE="goldenrod">Goldenrod
    <OPTION VALUE="lime">Lime
    <OPTION VALUE="powderblue">Powder Blue
    <OPTION VALUE="slategray">Slate Gray
</SELECT>
</TR>

<TR>
<TD>Select a text color:
<SELECT NAME="textColor">
    <OPTION VALUE="black">Black
    <OPTION VALUE="white">White
    <OPTION VALUE="navy">Navy Blue
    <OPTION VALUE="darkorange">Dark Orange
    <OPTION VALUE="seagreen">Sea Green
    <OPTION VALUE="teal">Teal
</SELECT>
</TR>

<TR>
<TD>Select "Welcome" heading font point size:
<SELECT NAME="h1Size">
    <OPTION VALUE="12">12
    <OPTION VALUE="14">14
    <OPTION VALUE="18">18
    <OPTION VALUE="24">24
    <OPTION VALUE="32">32
    <OPTION VALUE="48">48
</SELECT>
</TR>
</TABLE>
</FORM>
<DIV STYLE="position:absolute; left:200px; top:220px">
<BUTTON STYLE="width:80px" onClick="handleOK()">OK</BUTTON>  
<BUTTON STYLE="width:80px" onClick="handleCancel()">Cancel</BUTTON>
</DIV>
</BODY>
</HTML>
```

One last convenience feature of the dialog box window is the `onKeyPress` event handler in the text box. The function it invokes looks for the Enter key. If that key is pressed while the box has focus, the same `handleOK()` function is invoked, as if the user had clicked the OK button. This feature makes the dialog box behave as if the OK button is an automatic default, just as "real" dialog boxes.

You should observe several important structural changes that were made to turn the modal approach into a modeless one. Listing 16-41 shows the version of the main window modified for use with a modeless dialog box. Another global variable, `prefsDlog`, is initialized to eventually store the reference to the modeless window

returned by the `showModelessWindow()` method. The variable gets used to invoke the `init()` function inside the modeless dialog box, but also as conditions in an `if` construction surrounding the generation of the dialog box. The reason this is needed is to prevent multiple instances of the dialog box being created (the button is still alive while the modeless window is showing). The dialog box won't be created again as long as there is a value in `prefsDlog`, and the dialog box window has not been closed (picking up the `window.closed` property of the dialog box window).

The `showModelessDialog()` method's second parameter is a reference to the function in the main window that updates the main document. As you see in a moment, that function is invoked from the dialog box when the user clicks the OK or Apply buttons.

Listing 16-41: **Main Page for showModelessDialog()**

```
<HTML>
<HEAD>
<TITLE>window.setModelessDialog() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var currPrefs = new Array()
var prefsDlog
function getPrefsData() {
    if (!prefsDlog || prefsDlog.closed) {
        prefsDlog = showModelessDialog("lst16-42.htm", setPrefs,
        "dialogWidth:400px; dialogHeight:300px")
        prefsDlog.init(currPrefs)
    }
}

function setPrefs(prefs) {
    if (prefs["bgColor"]) {
        document.body.style.backgroundColor = prefs["bgColor"]
        currPrefs["bgColor"] = prefs["bgColor"]
    }
    if (prefs["textColor"]) {
        document.body.style.color = prefs["textColor"]
        currPrefs["textColor"] = prefs["textColor"]
    }
    if (prefs["h1Size"]) {
        document.all.welcomeHeader.style.fontSize = prefs["h1Size"]
        currPrefs["h1Size"] = prefs["h1Size"]
    }
    if (prefs["name"]) {
        document.all.firstName.innerText = prefs["name"]
        currPrefs["name"] = prefs["name"]
    }
}

function init() {
    document.all.firstName.innerText = "friend"
}
</SCRIPT>
```

Continued

Listing 16-41 *(continued)*

```
</HEAD>
<BODY BGCOLOR="#eeeeee" STYLE="margin:20px" onLoad="init()">
<H1>window.setModelessDialog() Method</H1>
<HR>
<H2 ID="welcomeHeader">Welcome, <SPAN ID="firstName"> </SPAN>!</H2>
<HR>
<P>Use this button to set style preferences for this page:
<BUTTON ID="prefsButton" onClick="getPrefsData()">
Preferences
</BUTTON>
</BODY>
</HTML>
```

Changes to the dialog box window document for a modeless version (Listing 16-42) are rather limited. A new button is added to the bottom of the screen for an Apply button. As in many dialog box windows you see in Microsoft products, the Apply button lets current settings in dialog boxes be applied to the current document but without closing the dialog box. This approach makes experimenting with settings easier.

The Apply button invokes a `handleApply()` function, which works the same as `handleOK()`, except the dialog box is not closed. But these two functions communicate back to the main window differently than a modal dialog box. The main window's processing function is passed as the second parameter of `showModelessDialog()` and is available as the `window.dialogArguments` property in the dialog box window's script. That function reference is assigned to a local variable in both functions, and the remote function is invoked, passing the results of the `getFormData()` function as parameter values back to the main window.

Listing 16-42: **Document for the Modeless Dialog Box**

```
<HTML>
<HEAD>
<TITLE>User Preferences</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// Close the dialog
function closeme() {
    window.close()
}

// Handle click of OK button
function handleOK() {
    var returnFunc = window.dialogArguments
    returnFunc(getFormData())
    closeme()
}
```

```
// Handle click of Apply button
function handleApply() {
    var returnFunc = window.dialogArguments
    returnFunc(getFormData())
}

// Handle click of Cancel button
function handleCancel() {
    window.returnValue = ""
    closeme()
}
// Generic function converts form element name-value pairs
// into an array
function getFormData() {
    var form = document.prefs
    var returnedData = new Array()
    // Harvest values for each type of form element
    for (var i = 0; i < form.elements.length; i++) {
        if (form.elements[i].type == "text") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else if (form.elements[i].type.indexOf("select") != -1) {
            returnedData[form.elements[i].name] =
            form.elements[i].options[form.elements[i].selectedIndex].value
        } else if (form.elements[i].type == "radio") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else if (form.elements[i].type == "checkbox") {
            returnedData[form.elements[i].name] = form.elements[i].value
        } else continue
    }
    return returnedData
}
// Initialize by setting form elements from passed data
function init(currPrefs) {
    if (currPrefs) {
        var form = document.prefs
        if (currPrefs["name"]) {
            form.name.value = currPrefs["name"]
        }
        if (currPrefs["bgColor"]) {
            setSelected(form.bgColor, currPrefs["bgColor"])
        }
        if (currPrefs["textColor"]) {
            setSelected(form.textColor, currPrefs["textColor"])
        }
        if (currPrefs["h1Size"]) {
            setSelected(form.h1Size, currPrefs["h1Size"])
        }
    }
}
// Utility function to set a SELECT element to one value
function setSelected(select, value) {
    for (var i = 0; i < select.options.length; i++) {
        if (select.options[i].value == value) {
```

Continued

Listing 16-42 *(continued)*

```
            select.selectedIndex = i
            break
        }
    }
    return
}
// Utility function to accept a press of the
// Enter key in the text field as a click of OK
function checkEnter() {
    if (window.event.keyCode == 13) {
        handleOK()
    }
}
</SCRIPT>
</HEAD>

<BODY BGCOLOR="#eeeeee" onLoad="init()">
<H2>Web Site Preferences</H2>
<HR>
<TABLE BORDER=0 CELLSPACING=2>
<FORM NAME="prefs" onSubmit="return false">
<TR>
<TD>Enter your first name:<INPUT NAME="name" TYPE="text" VALUE="" SIZE=20
onKeyDown="checkEnter()">
</TR>

<TR>
<TD>Select a background color:
<SELECT NAME="bgColor">
    <OPTION VALUE="beige">Beige
    <OPTION VALUE="antiquewhite">Antique White
    <OPTION VALUE="goldenrod">Goldenrod
    <OPTION VALUE="lime">Lime
    <OPTION VALUE="powderblue">Powder Blue
    <OPTION VALUE="slategray">Slate Gray
</SELECT>
</TR>

<TR>
<TD>Select a text color:
<SELECT NAME="textColor">
    <OPTION VALUE="black">Black
    <OPTION VALUE="white">White
    <OPTION VALUE="navy">Navy Blue
    <OPTION VALUE="darkorange">Dark Orange
    <OPTION VALUE="seagreen">Sea Green
    <OPTION VALUE="teal">Teal
</SELECT>
</TR>
```

```
<TR>
<TD>Select "Welcome" heading font point size:
<SELECT NAME="h1Size">
    <OPTION VALUE="12">12
    <OPTION VALUE="14">14
    <OPTION VALUE="18">18
    <OPTION VALUE="24">24
    <OPTION VALUE="32">32
    <OPTION VALUE="48">48
</SELECT>
</TR>
</TABLE>
</FORM>
<DIV STYLE="position:absolute; left:120px; top:220px">
<BUTTON STYLE="width:80px" onClick="handleOK()">OK</BUTTON>  
<BUTTON STYLE="width:80px" onClick="handleCancel()">Cancel</BUTTON>  
<BUTTON STYLE="width:80px" onClick="handleApply()">Apply</BUTTON>
</DIV>
</BODY>
</HTML>
```

The biggest design challenge you probably face with respect to these windows is deciding between a modal and modeless dialog box style. Some designers insist that modality has no place in a graphical user interface; others say that there are times when you need to focus the user on a very specific task before any further processing can take place. That's where a modal dialog box makes perfect sense.

sizeToContent()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) in NN6 to try the `sizeToContent()` method. Assuming that you are running The Evaluator from the `Chap13` directory on the CD-ROM (or the directory copied as-is to your hard disk), you can open a subwindow with one of the other files in the directory, and then size the subwindow. Enter the following statements into the top text box:

```
a = window.open("lst13-02.htm","")
a.sizeToContent()
```

The resized subwindow is at the minimum recommended width for a browser window, and at a height tall enough to display the little bit of content in the document.

Event handlers

onAfterPrint
onBeforePrint

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

The following script fragment assumes that the page includes a DIV element whose style sheet includes a setting of `display:none` as the page loads. Somewhere in the Head, the print-related event handlers are set as properties:

```
function showPrintCopyright() {
    document.all.printCopyright.style.display = "block"
}
function hidePrintCopyright() {
    document.all.printCopyright.style.display = "none"
}
window.onbeforeprint = showPrintCopyright
window.onafterprint = hidePrintCopyright
```

onBeforeUnload

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The simple page in Listing 16-43 shows you how to give the user a chance to stay on the page.

Listing 16-43: Using the onBeforeUnload Event Handler

```
<HTML>
<HEAD>
<TITLE>onBeforeUnload Event Handler</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function verifyClose() {
    event.returnValue = "We really like you and hope you will stay longer."
}
```

```
window.onbeforeunload = verifyClose
</SCRIPT>

</HEAD>
<BODY>
<H1>onBeforeUnload Event Handler</H1>
<HR>
<P>Use this button to navigate to the previous page:
<BUTTON ID="go" onClick="history.back()">
Go Back
</BUTTON>
</BODY>
</HTML>
```

onHelp

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The following script fragment can be embedded in the IE5-only modeless dialog box code in Listing 16-44 to provide context-sensitive help within the dialog box. Help messages for only two of the form elements are shown here, but in a real application you add messages for the rest.

```
function showHelp() {
    switch (event.srcElement.name) {
        case "bgColor" :
            alert("Choose a color for the main window\'s background.")
            break
        case "name" :
            alert("Enter your first name for a friendly greeting.")
            break
        default :
            alert("Make preference settings for the main page styles.")
    }
    event.returnValue = false
}
window.onhelp = showHelp
```

Because this page's help focuses on form elements, the `switch` construction cases are based on the `name` properties of the form elements. For other kinds of pages, the `id` properties may be more appropriate.

FRAME Element Object

Properties

borderColor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Although you may experience problems (especially in IE5) changing the color of a single frame border, the W3C DOM syntax would look like the following if the script were inside the framesetting document:

```
document.getElementById("contentsFrame").borderColor = "red"
```

The IE-only version would be:

```
document.all["contentsFrame"].borderColor = "red"
```

These examples assume the frame name arrives to a script function as a string. If the script is executing in one of the frames of the frameset, add a reference to `parent` in the preceding statements.

contentDocument

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

A framesetting document script might be using the ID of a FRAME element to read or adjust one of the element properties, and then need to perform some action on the content of the page through its `document` object. You can get the reference to the `document` object via a statement, such as the following:

```
var doc = document.getElementById("FRAME3").contentDocument
```

Then your script can, for example, dive into a form in the document:

```
var val = doc.mainForm.entry.value
```

Document

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

While you have far easier ways to reach the `document` object of another frame (`parent.`*`otherFrameName`*`.document`), the following statement takes the long way to get there to retrieve the number of forms in the document of another frame:

```
var formCount = parent.document.all.contentsFrame.Document.forms.length
```

Using the `Document` property only truly makes sense when a function is passed a FRAME or IFRAME element object reference as a parameter, and the script must, among other things more related to those objects, access the document contained by those elements.

frameBorder

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The default value for the `frameBorder` property is `yes`. You can use this setting to create a toggle script (which, unfortunately, does not change the appearance in IE). The W3C-compatible version looks like the following:

```
function toggleFrameScroll(frameID) {
    var theFrame = document.getElementById(frameID)
    if (theFrame.frameBorder == "yes") {
        theFrame.frameBorder = "no"
    } else {
        theFrame.frameBorder    = "yes"
    }
}
```

height
width

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The following fragment assumes a frameset defined with two frames set up as two columns within the frameset. The statements here live in the framesetting document. They retrieve the current width of the left frame and increase the width of that frame by ten percent. Syntax shown here is for the W3C DOM, but can be easily adapted to IE-only terminology.

```
var frameWidth = document.getElementById("leftFrame").width
document.getElementById("mainFrameset").cols = (Math.round(frameWidth * 1.1)) +
",*"
```

Notice how the numeric value of the existing frame width is first increased by ten percent and then concatenated to the rest of the string property assigned to the frameset's `cols` property. The asterisk after the comma means that the browser should figure out the remaining width and assign it to the right-hand frame.

noResize

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statement turns off the ability for a frame to be resized:

```
parent.document.getElementById("myFrame1").noResize = true
```

Because of the negative nature of the property name, it may be difficult to keep the logic straight (setting `noResize` to `true` means that resizability is turned off). Keep a watchful eye on your Boolean values.

scrolling

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listing 16-45 produces a frameset consisting of eight frames. The content for the frames is generated by a script within the frameset (via the `fillFrame()` function). Event handlers in the Body of each frame invoke the `toggleFrameScroll()` function. Both ways of referencing the FRAME element object are shown, with the IE-only version commented out.

In the `toggleFrameScroll()` function, the `if` condition checks whether the property is set to something other than `no`. This allows the condition to evaluate to `true` if the property is set to either `auto` (the first time) or `yes` (as set by the function). Note that the scrollbars don't disappear from the frames in IE5.5 or NN6.

Listing 16-45: **Controlling the FRAME.scrolling Property**

```
<HTML>
<HEAD>
<TITLE>frame.scrolling Property</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
function toggleFrameScroll(frameID) {
    // IE5 & NN6 version
    var theFrame = document.getElementById(frameID)
    // IE4+ version
    // var theFrame = document.all[frameID]

    if (theFrame.scrolling != "no") {
        theFrame.scrolling = "no"
    } else {
        theFrame.scrolling = "yes"
    }
}
// generate content for each frame
function fillFrame(frameID) {
    var page = "<HTML><BODY onClick='parent.toggleFrameScroll(\"" +
        frameID + "\")'><SPAN STYLE='font-size:24pt'>"
    page += "<P>This frame has the ID of:</P><P>" + frameID + ".</P>"
    page += "</SPAN></BODY></HTML>"
    return page
}
</SCRIPT>
<FRAMESET ID="outerFrameset" COLS="50%,50%">
    <FRAMESET ID="innerFrameset1" ROWS="25%,25%,25%,25%">
        <FRAME ID="myFrame1" SRC="javascript:parent.fillFrame('myFrame1')">
        <FRAME ID="myFrame2" SRC="javascript:parent.fillFrame('myFrame2')">
        <FRAME ID="myFrame3" SRC="javascript:parent.fillFrame('myFrame3')">
        <FRAME ID="myFrame4" SRC="javascript:parent.fillFrame('myFrame4')">
    </FRAMESET>
    <FRAMESET ID="innerFrameset2" ROWS="25%,25%,25%,25%">
        <FRAME ID="myFrame5" SRC="javascript:parent.fillFrame('myFrame5')">
        <FRAME ID="myFrame6" SRC="javascript:parent.fillFrame('myFrame6')">
        <FRAME ID="myFrame7" SRC="javascript:parent.fillFrame('myFrame7')">
        <FRAME ID="myFrame8" SRC="javascript:parent.fillFrame('myFrame8')">
    </FRAMESET>
</FRAMESET>
</HTML>
```

src

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

For best results, use fully formed URLs as value for the `src` property, as shown here:

```
parent.document.getElementById("mainFrame").src = "http://www.dannyg.com"
```

Relative URLs and `javascript:` pseudo-URLs will also work most of the time.

FRAMESET Element Object

Properties

border

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Even though the property is read/write in IE4+, changing the value does not change the thickness of the border you see in the browser. If you need to find the thickness of the border, a script reference from one of the frame's documents would look like the following:

```
var thickness = parent.document.all.outerFrameset.border
```

borderColor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

To retrieve the current color setting in a frameset, a script reference from one of the frame's documents would look like the following:

```
var borderColor = parent.document.all.outerFrameset.borderColor
```

cols
rows

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listings 16-46 through 16-48 show the HTML for a frameset and two of the three documents that go into the frameset. The final document is an HTML version of the U.S. Bill of Rights, which is serving here as a content frame for the demonstration.

The frameset listing (16-46) shows a three-frame setup. Down the left column is a table of contents (16-47). The right column is divided into two rows. In the top row is a simple control (16-48) that hides and shows the table of contents frame. As the user clicks the hot text of the control (located inside a SPAN element), the `onClick` event handler invokes the `toggleTOC()` function in the frameset. Figure 2-10 shows the frameset with the menu exposed.

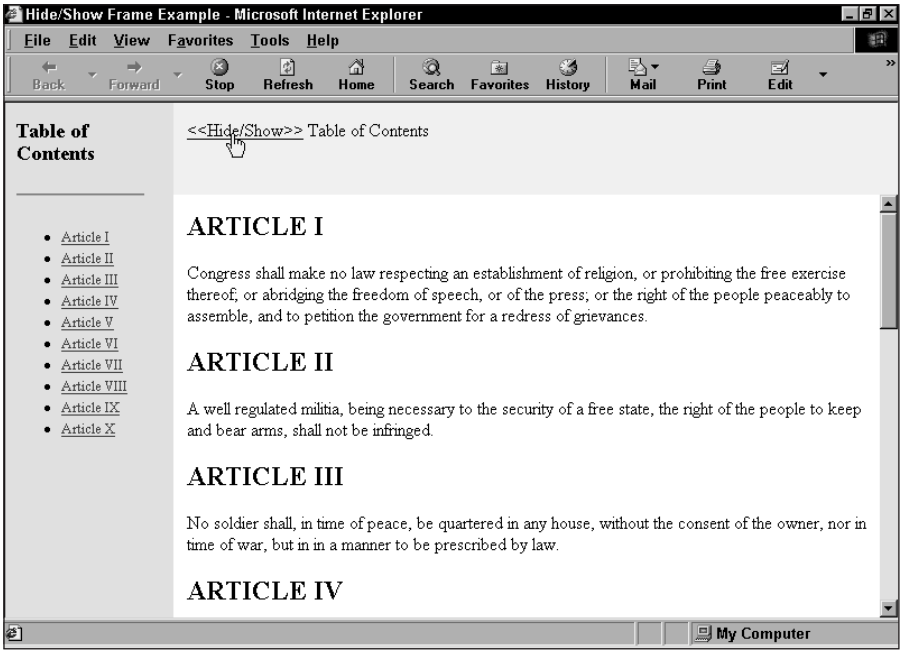


Figure 2-10: Frameset specifications are modified on the fly when you click on the top control link.

Syntax used in this example is W3C-compatible. To modify this for IE-only, you replace `document.getElementById("outerFrameset")` with `document.all.outerFrameset` and `elem.firstChild.nodeValue` to `elem.innerText`. You can also branch within the scripts to accommodate both styles.

Listing 16-46: Frameset and Script for Hiding/Showing a Frame

```
<HTML>
<HEAD>
<TITLE>Hide/Show Frame Example</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
var origCols
function toggleTOC(elem, frm) {
    if (origCols) {
        showTOC(elem)
    } else {
        hideTOC(elem, frm)
    }
}
function hideTOC(elem, frm) {
    var frameset = document.getElementById("outerFrameset")
    origCols = frameset.cols
    frameset.cols = "0,*"
}
function showTOC(elem) {
    if (origCols) {
        document.getElementById("outerFrameset").cols = origCols
        origCols = null
    }
}
</SCRIPT>
<FRAMESET ID="outerFrameset" FRAMEBORDER="no" COLS="150,*">
    <FRAME ID="TOC" NAME="TOCFrame" SRC="lst16-47.htm">
    <FRAMESET ID="innerFrameset1" ROWS="80,*">
        <FRAME ID="controls" NAME="controlsFrame" SRC="lst16-48.htm">
        <FRAME ID="content" NAME="contentFrame" SRC="bofright.htm">
    </FRAMESET>
</FRAMESET>
</HTML>
```

When a user clicks the hot spot to hide the frame, the script copies the original `cols` property settings to a global variable. The variable is used in `showTOC()` to restore the frameset to its original proportions. This allows a designer to modify the HTML for the frameset without also having to dig into scripts to hard-wire the restored size.

Listing 16-47: **Table of Contents Frame Content**

```
<HTML>
<HEAD>
<TITLE>Table of Contents</TITLE>
</HEAD>
<BODY BGCOLOR="#eeeeee">
<H3>Table of Contents</H3>
<HR>
<UL STYLE="font-size:10pt">
<LI><A HREF="bofright.htm#article1" TARGET="contentFrame">Article I</A></LI>
<LI><A HREF="bofright.htm#article2" TARGET="contentFrame">Article II</A></LI>
<LI><A HREF="bofright.htm#article3" TARGET="contentFrame">Article III</A></LI>
<LI><A HREF="bofright.htm#article4" TARGET="contentFrame">Article IV</A></LI>
<LI><A HREF="bofright.htm#article5" TARGET="contentFrame">Article V</A></LI>
<LI><A HREF="bofright.htm#article6" TARGET="contentFrame">Article VI</A></LI>
<LI><A HREF="bofright.htm#article7" TARGET="contentFrame">Article VII</A></LI>
<LI><A HREF="bofright.htm#article8" TARGET="contentFrame">Article VIII</A></LI>
<LI><A HREF="bofright.htm#article9" TARGET="contentFrame">Article IX</A></LI>
<LI><A HREF="bofright.htm#article10" TARGET="contentFrame">Article X</A></LI>
</UL>
</BODY>
</HTML>
```

Listing 16-48: **Control Panel Frame**

```
<HTML>
<HEAD>
<TITLE>Control Panel</TITLE>
</HEAD>
<BODY>
<P>
<SPAN ID="tocToggle"
      STYLE="text-decoration:underline; cursor:hand"
      onClick="parent.toggleTOC(this)"> &lt;&lt;Hide/Show&gt;&gt;</SPAN>
Table of Contents
</P>
</BODY>
</HTML>
```

frameBorder

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The default value for the `frameBorder` property is `yes`. You can use this setting to create a toggle script (which, unfortunately, does not change the appearance in IE). The IE4+-compatible version looks like the following:

```
function toggleFrameScroll(framesetID) {
    var theFrameset = document.all(framesetID)
    if (theFrameset.frameBorder == "yes") {
        theFrameset.frameBorder = "no"
    } else {
        theFrameset.frameBorder = "yes"
    }
}
```

frameSpacing

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Even though the property is read/write in IE4+, changing the value does not change the thickness of the frame spacing you see in the browser. If you need to find the spacing as set by the tag's attribute, a script reference from one of the frame's documents would look like the following:

```
var spacing = parent.document.all.outerFrameset.frameSpacing
```

IFRAME Element Object

Properties

align

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The default setting for an IFRAME alignment is `baseline`. A script can shift the IFRAME to be flush with the right edge of the containing element as follows:

```
document.getElementById("iframe1").align = "right"
```

contentDocument

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

A document script might be using the ID of an IFRAME element to read or adjust one of the element properties; it then needs to perform some action on the content of the page through its `document` object. You can get the reference to the `document` object via a statement, such as the following:

```
var doc = document.getElementById("FRAME3").contentDocument
```

Then your script can, for example, dive into a form in the document:

```
var val = doc.mainForm.entry.value
```

frameBorder

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See the example for the `FRAME.frameBorder` property earlier in this chapter.

hspace
vspace

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The following fragment sets the white space surrounding an IFRAME element to an equal amount:

```
document.all.myIframe.hspace = 20
document.all.myIframe.vspace = 20
```

Unfortunately these changes do not work for IE5/Windows.

scrolling

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following `toggleIFrameScroll()` function accepts a string of the IFRAME element's ID as a parameter and switches between on and off scroll bars in the IFRAME. The `if` condition checks whether the property is set to something other than `no`. This test allows the condition to evaluate to true `if` the property is set to either `auto` (the first time) or `yes` (as set by the function).

```
function toggleFrameScroll(frameID) {
    // IE5 & NN6 version
    var theFrame = document.getElementById(frameID)
    // IE4+ version
    // var theFrame = document.all[frameID]
    if (theFrame.scrolling != "no") {
        theFrame.scrolling = "no"
    } else {
        theFrame.scrolling = "yes"
    }
}
```

src

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

For best results, use fully formed URLs as value for the `src` property, as shown here:

```
document.getElementById("myIframe").src = "http://www.dannyg.com"
```

Relative URLs and `javascript:` pseudo-URLs also work most of the time.

popup Object

Properties

document

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `popup` object and its properties. Enter the following statements into the top text box. The first statement creates a pop-up window, whose reference is assigned to the `a` global variable. Next, a reference to the body of the pop-up's document is preserved in the `b` variable for the sake of convenience. Further statements work with these two variables.

```
a = window.createPopup()
b = a.document.body
b.style.border = "solid 2px black"
b.style.padding = "5px"
b.innerHTML = "<P>Here is some text in a popup window</P>"
a.show(200,100, 200, 50, document.body)
```

See the description of the `show()` method for details on the parameters.

isOpen

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `isOpen` property. Enter the following statements into the top text box. The sequence begins with a creation of a simple pop-up window, whose reference is assigned to the `a` global variable. Note that the final statement is actually two statements, which are designed so that the second statement executes while the pop-up window is still open.

```
a = window.createPopup()
a.document.body.innerHTML = "<P>Here is a popup window</P>"
a.show(200,100, 200, 50, document.body); alert("Popup is open:" + a.isOpen)
```

If you then click into the main window to hide the pop-up, you will see a different result if you enter the following statement into the top text box by itself:

```
alert("Popup is open:" + a.isOpen)
```

Methods

hide()
show(*left, top, width, height*[, *positioningElementRef*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

Listing 16-49 demonstrates both the `show()` and `hide()` methods for a `popup` object. A click of the button on the page invokes the `selfTimer()` function, which acts as the main routine for this page. The goal is to produce a pop-up window that "self-destructs" five seconds after it appears. Along the way, a message in the pop-up counts down the seconds.

A reference to the pop-up window is preserved as a global variable, called `popup`. After the `popup` object is created, the `initContent()` function stuffs the content into the pop-up by way of assigning `style` properties and some `innerHTML` for the body of the document that is automatically created when the pop-up is generated. A SPAN element is defined so that another function later on can modify the content of just that segment of text in the pop-up. Notice that the assignment of content to the pop-up is predicated on the pop-up window having been initialized (by virtue of the `popup` variable having a value assigned to it) and that the pop-up window is not showing. While invoking `initContent()` under any other circumstances is probably impossible, the validation of the desired conditions is good programming practice.

Back in `selfTimer()`, the `popup` object is displayed. Defining the desired size requires some trial and error to make sure the pop-up window comfortably accommodates the text that is put into the pop-up in the `initContent()` function.

With the pop-up window showing, now is the time to invoke the `countDown()` function. Before the function performs any action, it validates that the pop-up has been initialized and is still visible. If a user clicks the main window while the counter is counting down, this changes the value of the `isOpen` property to `false`, and nothing inside the `if` condition executes.

This `countDown()` function grabs the inner text of the SPAN and uses `paresInt()` to extract just the integer number (using base 10 numbering, because we're dealing with zero-leading numbers that can potentially be regarded as octal values). The condition of the `if` construction decreases the retrieved integer by one. If the decremented value is zero, then the time is up, and the pop-up window is

hidden with the `popup` global variable returned to its original, `null` value. But if the value is other than zero, then the inner text of the SPAN is set to the decremented value (with a leading zero), and the `setTimeout()` method is called upon to reinvoke the `countDown()` function in one second (1000 milliseconds).

Listing 16-49: **Hiding and Showing a Pop-up**

```
<HTML>
<HEAD>
<TITLE>popup Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var popup
function initContent() {
    if (popup && !popup.isOpen) {
        var popBody = popup.document.body
        popBody.style.border = "solid 3px red"
        popBody.style.padding = "10px"
        popBody.style.fontSize = "24pt"
        popBody.style.textAlign = "center"
        var bodyText = "<P>This popup will self-destruct in "
        bodyText += "<SPAN ID='counter'>05</SPAN>"
        bodyText += " seconds...</P>"
        popBody.innerHTML = bodyText
    }
}
function countDown() {
    if (popup && popup.isOpen) {
        var currCount = parseInt(popup.document.all.counter.innerText, 10)
        if (--currCount == 0) {
            popup.hide()
            popup = null
        } else {
            popup.document.all.counter.innerText = "0" + currCount
            setTimeout("countDown()", 1000)
        }
    }
}
function selfTimer() {
    popup = window.createPopup()
    initContent()
    popup.show(200,200,400,100,document.body)
    setTimeout("countDown()", 1000)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Impossible Mission" onClick="selfTimer()">
</FORM>
</BODY>
</HTML>
```

The `hide()` method here is invoked by a script that is running while the pop-up window is showing. Because a pop-up window automatically goes away if a user clicks the main window, it is highly unlikely that the `hide()` method would ever be invoked by itself in response to user action in the main window. If you want a script in the pop-up window to close the pop-up, use `parentWindow.close()`.

CHAPTER 3

Location and History Objects (Chapter 17)

In This Chapter

Loading new pages and other media types via the `location` object

Passing data between pages via URLs

Navigating through the browser history under script control

While both the `location` and `history` objects contain valuable information about the user's Web surfing habits and even the content of forms, they could also be abused by nefarious scripts that wish to invade the privacy of unsuspecting site visitors. As a result, browsers do not expose the private details to scripts (except in NN4+ via signed scripts and the user's express permission).

The `location` object, however, is still an important object to know and exploit. As shown in the examples here, you can use it as one cookie-free way to pass text data from one page to another. And the object remains the primary way scripts load a new page into the browser.

Examples Highlights

- The frameset listing for the `location.host` property demonstrates several `location` object properties. You also find an example of how signed scripts can be used in NN4+ to access `location` object properties for pages served by a different domain.
- Listings for the `location.search` property pass data from one page to another via a URL. In this case, a script in a page not only makes sure that your site gets served within the prescribed frameset, but the specific page also gets loaded into one of the frames, even if it is not the page specified in the frameset's definition.
- Observe the `location.replace()` method's example. This method comes in handy when you don't want one of your pages to become part of the browser's history: Clicking the Back button skips over the replaced page.
- Run Listings 17-12 and 17-13 for the `history.back()` method to see how the behavior of this method varies among browsers. Consult the *JavaScript Bible* text for details on the evolution of this method.

Location Object

Properties

hash

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

When you load the script in Listing 17-1, adjust the size of the browser window so only one section is visible at a time. When you click a button, its script navigates to the next logical section in the progression and eventually takes you back to the top.

Listing 17-1: A Document with Anchors

```
<HTML>
<HEAD>
<TITLE>location.hash Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function goNextAnchor(where) {
    window.location.hash = where
}
</SCRIPT>
</HEAD>

<BODY>

<A NAME="start"><H1>Top</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec1')">
</FORM>
<HR>
<A NAME="sec1"><H1>Section 1</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec2')">
</FORM>
<HR>
<A NAME="sec2"><H1>Section 2</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec3')">
</FORM>
<HR>
<A NAME="sec3"><H1>Section 3</H1></A>
<FORM>
```

```
<INPUT TYPE="button" NAME="next" VALUE="BACK TO TOP"
onClick="goNextAnchor('start')">
</FORM>

</BODY>
</HTML>
```

Anchor names are passed as parameters with each button's `onClick` event handler. Instead of going through the work of assembling a `window.location` value in the function by appending a literal hash mark and the value for the anchor, here I simply modify the `hash` property of the current window's location. This is the preferred, cleaner method.

If you attempt to read back the `window.location.hash` property in an added line of script, however, the window's actual URL probably will not have been updated yet, and the browser will appear to be giving your script false information. To prevent this problem in subsequent statements of the same function, construct the URLs of those statements from the same variable values you use to set the `window.location.hash` property — don't rely on the browser to give you the values you expect.

host

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Use the documents in Listings 17-2 through 17-4 as tools to help you learn the values that the various `window.location` properties return. In the browser, open the file for Listing 17-2. This file creates a two-frame window. The left frame contains a temporary placeholder (Listing 17-4) that displays some instructions. The right frame has a document (Listing 17-3) that enables you to load URLs into the left frame and get readings on three different windows available: the parent window (which creates the multiframe window), the left frame, and the right frame.

Listing 17-2: **Frameset for the Property Picker**

```
<HTML>
<HEAD>
<TITLE>window.location Properties</TITLE>
</HEAD>
<FRAMESET COLS="50%,50%" BORDER=1 BORDERCOLOR="black">
    <FRAME NAME="Frame1" SRC="lst17-04.htm">
    <FRAME NAME="Frame2" SRC="lst17-03.htm">
</FRAMESET>
</HTML>
```

Listing 17-3: **Property Picker**

```
<HTML>
<HEAD>
<TITLE>Property Picker</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var isNav4 = (navigator.appName == "Netscape" &&
navigator.appVersion.charAt(0) >= 4) ? true : false

function fillLeftFrame() {
    newURL = prompt("Enter the URL of a document to show in the left frame:","")
    if (newURL != null && newURL != "") {
    parent.frames[0].location = newURL
    }
}

function showLocationData(form) {
    for (var i = 0; i <3; i++) {
        if (form.whichFrame[i].checked) {
            var windName = form.whichFrame[i].value
            break
        }
    }
    var theWind = "" + windName + ".location"
    if (isNav4) {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead")
    }
    var theObj = eval(theWind)
    form.windName.value = windName
    form.windHash.value = theObj.hash
    form.windHost.value = theObj.host
    form.windHostname.value = theObj.hostname
    form.windHref.value = theObj.href
    form.windPath.value = theObj.pathname
    form.windPort.value = theObj.port
    form.windProtocol.value = theObj.protocol
    form.windSearch.value = theObj.search
    if (isNav4) {
    netscape.security.PrivilegeManager.disablePrivilege("UniversalBrowserRead")
    }
}
</SCRIPT>
</HEAD>
<BODY>
Click the "Open URL" button to enter the location of an HTML document to display
in the left frame of this window.
<FORM>
<INPUT TYPE="button" NAME="opener" VALUE="Open URL..."
onClick="fillLeftFrame()">
<HR>
<CENTER>
Select a window/frame. Then click the "Show Location Properties" button to view
each window.location property value for the desired window.<P>
```

```
<INPUT TYPE="radio" NAME="whichFrame" VALUE="parent" CHECKED>Parent window
<INPUT TYPE="radio" NAME="whichFrame" VALUE="parent.frames[0]">Left frame
<INPUT TYPE="radio" NAME="whichFrame" VALUE="parent.frames[1]">This frame
<P>
<INPUT TYPE="button" NAME="getProperties" VALUE="Show Location Properties"
onClick="showLocationData(this.form)">
<INPUT TYPE="reset" VALUE="Clear"><P>
<TABLE BORDER=2>
<TR><TD ALIGN=right>Window:</TD><TD><INPUT TYPE="text" NAME="windName"
SIZE=30></TD></TR>
<TR><TD ALIGN=right>hash:</TD>
<TD><INPUT TYPE="text" NAME="windHash" SIZE=30></TD></TR>

<TR><TD ALIGN=right>host:</TD>
<TD><INPUT TYPE="text" NAME="windHost" SIZE=30></TD></TR>

<TR><TD ALIGN=right>hostname:</TD>
<TD><INPUT TYPE="text" NAME="windHostname" SIZE=30></TD></TR>

<TR><TD ALIGN=right>href:</TD>
<TD><TEXTAREA NAME="windHref" ROWS=3 COLS=30 WRAP="soft">
</TEXTAREA></TD></TR>

<TR><TD ALIGN=right>pathname:</TD>
<TD><TEXTAREA NAME="windPath" ROWS=3 COLS=30 WRAP="soft">
</TEXTAREA></TD></TR>

<TR><TD ALIGN=right>port:</TD>
<TD><INPUT TYPE="text" NAME="windPort" SIZE=30></TD></TR>

<TR><TD ALIGN=right>protocol:</TD>
<TD><INPUT TYPE="text" NAME="windProtocol" SIZE=30></TD></TR>

<TR><TD ALIGN=right>search:</TD>
<TD><TEXTAREA NAME="windSearch" ROWS=3 COLS=30 WRAP="soft">
</TEXTAREA></TD></TR>
</TABLE>
</CENTER>
</FORM>
</BODY>
</HTML>
```

Listing 17-4: **Placeholder Document for Listing 17-2**

```
<HTML>
<HEAD>
<TITLE>Opening Placeholder</TITLE>
```

Continued

Listing 17-4 *(continued)*

```
</HEAD>
<BODY>
Initial placeholder. Experiment with other URLs for this frame (see right).
</BODY>
</HTML>
```

Figure 3-1 shows the dual-frame browser window with the left frame loaded with a page from my Web site.

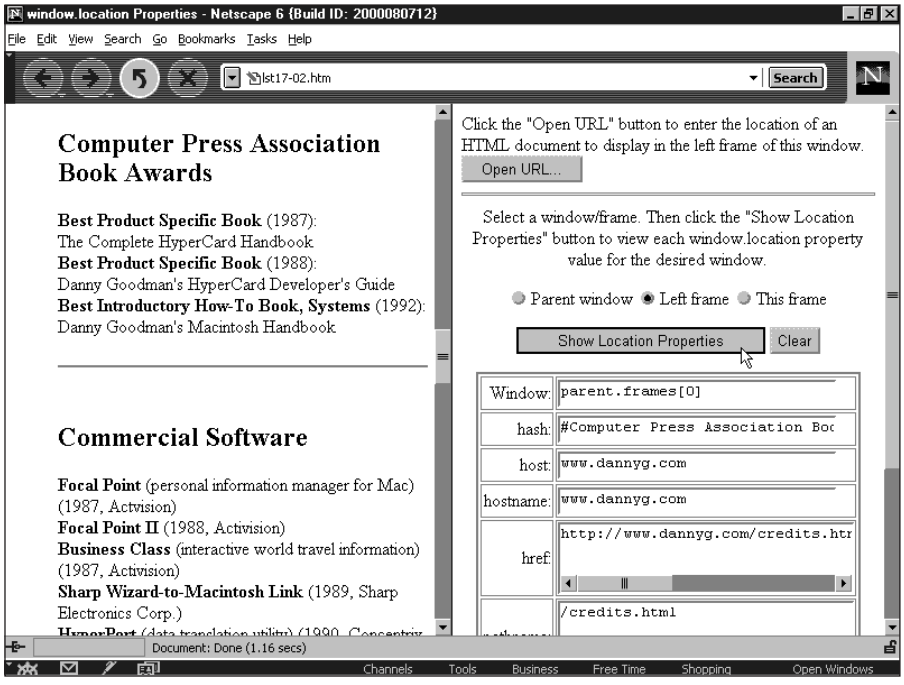


Figure 3-1: Browser window loaded to investigate window.location properties

For the best results, open a URL to a Web document on the network from the same domain and server from which you load the listings (perhaps your local hard disk). If possible, load a document that includes anchor points to navigate through a long document. Click the Left frame radio button, and then click the button that shows all properties. This action fills the table in the right frame with all the available `location` properties for the selected window. Figure 3-2 shows the complete results for a page from my Web site that is set to an anchor point.

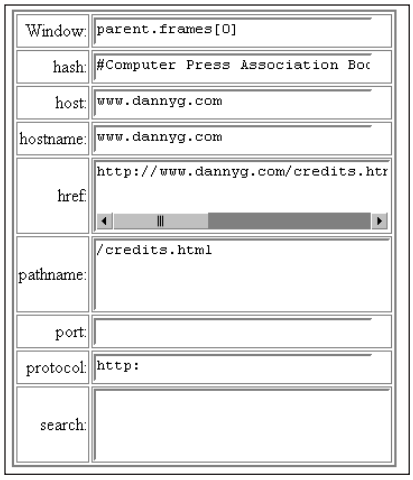


Figure 3-2: Readout of all window.location properties for the left frame

Attempts to retrieve these properties from URLs outside of your domain and server result in a variety of responses based on your browser and browser version. NN2 returns `null` values for all properties. NN3 presents an "access disallowed" security alert. With codebase principals turned on in NN4 (see Chapter 46 of the *JavaScript Bible*), the proper values appear in their fields. IE3 does not have the same security restrictions that Navigator does, so all values appear in their fields. But in IE4+, you get a "permission denied" error alert. See the following discussion for the meanings of the other listed properties and instructions on viewing their values.

hostname

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listings 17-2 through 17-4 for a set of related pages to help you view the hostname data for a variety of other pages.

href

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 17-5 includes the `unescape()` function in front of the part of the script that captures the URL. This function serves cosmetic purposes by displaying the pathname in alert dialog boxes for browsers that normally display the ASCII-encoded version.

Listing 17-5: Extracting the Directory of the Current Document

```
<HTML>
<HEAD>
<TITLE>Extract pathname</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// general purpose function to extract URL of current directory
function getDirPath(URL) {
    var result = unescape(URL.substring(0,(URL.lastIndexOf("/")) + 1))
    return result
}
// handle button event, passing work onto general purpose function
function showDirPath(URL) {
    alert(getDirPath(URL))
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button" VALUE="View directory URL"
onClick="showDirPath(window.location.href)">
</FORM>
</BODY>
</HTML>
```

pathname

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listings 17-2 through 17-4 earlier in this chapter for a multiple-frame example you can use to view the `location.pathname` property for a variety of URLs of your choice.

port

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

If you have access to URLs containing port numbers, use the documents in Listings 17-2 through 17-4 to experiment with the output of the `location.port` property.

protocol

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listings 17-2 through 17-4 for a multiple-frame example you can use to view the `location.protocol` property for a variety of URLs. Also try loading an FTP site to see the `location.protocol` value for that type of URL.

search

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

As mentioned in the opening of Chapter 16 of the *JavaScript Bible* about frames, you can force a particular HTML page to open inside the frameset for which it is designed. But with the help of the search string, you can reuse the same framesetting document to accommodate any number of content pages that go into one of the frames (rather than specifying a separate frameset for each possible combination of pages in the frameset). The listings in this section create a simple example of how to force a page to load in a frameset by passing some information about the page to the frameset. Thus, if a user has a URL to one of the content frames (perhaps it has been bookmarked by right-clicking the frame or it comes up as a search engine result), the page appears in its designated frameset the next time the user visits the page.

The fundamental task going on in this scheme has two parts. The first is in each of the content pages where a script checks whether the page is loaded inside a frameset. If the frameset is missing, then a search string is composed and appended

to the URL for the framesetting document. The framesetting document has its own short script that looks for the presence of the search string. If the string is there, then the script extracts the search string data and uses it to load that specific page into the content frame of the frameset.

Listing 17-6 is the framesetting document. The `getSearchAsArray()` function is more complete than necessary for this simple example, but you can use it in other instances to convert any number of name/value pairs passed in the search string (in traditional format of `name1=value1&name2=value2&etc.`) into an array whose indexes are the names (making it easier for scripts to extract a specific piece of passed data). Version branching takes place because, for convenience, the `getSearchAsArray()` function uses text and array methods that don't exist in browsers prior to NN3 or IE4.

Listing 17-6: **A Smart Frameset**

```
<HTML>
<HEAD>
<TITLE>Example Frameset</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// Convert location.search into an array of values
// indexed by name.
function getSearchAsArray() {
    var minNav3 = (navigator.appName == "Netscape" &&
parseInt(navigator.appVersion) >= 3)
    var minIE4 = (navigator.appName.indexOf("Microsoft") >= 0 &&
parseInt(navigator.appVersion) >= 4)
    var minDOM = minNav3 || minIE4   // baseline DOM required for this function
    var results = new Array()
    if (minDOM) {
        var input = unescape(location.search.substr(1))
        if (input) {
            var srchArray = input.split("&")
            var tempArray = new Array()
            for (var i = 0; i < srchArray.length; i++) {
                tempArray = srchArray[i].split("=")
                results[tempArray[0]] = tempArray[1]
            }
        }
    }
    return results
}
function loadFrame() {
    if (location.search) {
        var srchArray = getSearchAsArray()
        if (srchArray["content"]) {
            self.content.location.href = srchArray["content"]
        }
    }
}
</SCRIPT>
</HEAD>
```

```
<FRAMESET COLS="250,*" onLoad="loadFrame()">
    <FRAME NAME="toc" SRC="lst17-07.htm">
    <FRAME NAME="content" SRC="lst17-08.htm">
</FRAMESET>
</HTML>
```

Listing 17-7 is the HTML for the table of contents frame. Nothing elaborate goes on here, but you can see how normal navigation works for this simplified frameset.

Listing 17-7: **The Table of Contents**

```
<HTML>
<HEAD>
<TITLE>Table of Contents</TITLE>
</HEAD>
<BODY BGCOLOR="#eeeeee">
<H3>Table of Contents</H3>
<HR>
<UL>
<LI><A HREF="lst17-08.htm" TARGET="content">Page 1</A></LI>
<LI><A HREF="lst17-08a.htm" TARGET="content">Page 2</A></LI>
<LI><A HREF="lst17-08b.htm" TARGET="content">Page 3</A></LI>
</UL>
</BODY>
</HTML>
```

Listing 17-8 shows one of the content pages. As the page loads, the `checkFrameset()` function is invoked. If the window does not load inside a frameset, then the script navigates to the framesetting page, passing the current content URL as a search string. Notice that for browsers that support the `location.replace()` method, the loading of this page on its own does not get recorded to the browser's history and isn't accessed if the user hits the Back button.

Listing 17-8: **A Content Page**

```
<HTML>
<HEAD>
<TITLE>Page 1</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function checkFrameset() {
    var minNav3 = (navigator.appName == "Netscape" &&
parseInt(navigator.appVersion) >= 3)
    var minIE4 = (navigator.appName.indexOf("Microsoft") >= 0 &&
parseInt(navigator.appVersion) >= 4)
```

Continued

Listing 17-8 *(continued)*

```
    var minDOM = minNav3 || minIE4   // baseline DOM required for this function
    var isNav4 = (navigator.appName == "Netscape" &&
parseInt(navigator.appVersion) == 4)
    if (parent == window) {
        // Don't do anything if running NN4
        // so that the frame can be printed on its own
        if (isNav4 && window.innerWidth == 0) {
            return
        }
        if (minDOM) {
            // Use replace() to keep current page out of history
            location.replace("lst17-06.htm?content=" + escape(location.href))
        } else {
            location.href = " lst17-06.htm?content=" + escape(location.href)
        }
    }
}
// Invoke the function
checkFrameset()
</SCRIPT>
</HEAD>
<BODY>
<H1>Page 1</H1>
<HR>
</BODY>
</HTML>
```

In practice, I recommend placing the code for the `checkFrameset()` function and call to it inside an external `.js` library and linking that library into each content document of the frameset. That's why the function assigns the generic `location.href` property to the search string — you can use it on any content page.

Methods

`reload(`*`unconditionalGETBoolean`*`)`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

To experience the difference between the two loading styles, load the document in Listing 17-9. Click a radio button, enter some new text, and make a choice in the SELECT object. Clicking the Soft Reload/Refresh button invokes a method that

reloads the document as if you had clicked the browser's Reload/Refresh button. It also preserves the visible properties of form elements. The Hard Reload button invokes the `location.reload()` method, which resets all objects to their default settings.

Listing 17-9: **Hard versus Soft Reloading**

```
<HTML>
<HEAD>
<TITLE>Reload Comparisons</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
function hardReload() {
    location.reload(true)
}
function softReload() {
    history.go(0)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myForm">
<INPUT TYPE="radio" NAME="rad1" VALUE = 1>Radio 1<BR>
<INPUT TYPE="radio" NAME="rad1" VALUE = 2>Radio 2<BR>
<INPUT TYPE="radio" NAME="rad1" VALUE = 3>Radio 3<P>
<INPUT TYPE="text" NAME="entry" VALUE="Original"><P>
<SELECT NAME="theList">
<OPTION>Red
<OPTION>Green
<OPTION>Blue
</SELECT>
<HR>
<INPUT TYPE="button" VALUE="Soft Reload" onClick="softReload()">
<INPUT TYPE="button" VALUE="Hard Reload" onClick="hardReload()">
</FORM>
</BODY>
</HTML>
```

`replace("URL")`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

Calling the `location.replace()` method navigates to another URL similarly to assigning a URL to the location. The difference is that the document doing the calling

doesn't appear in the history list after the new document loads. Check the history listing (in your browser's usual spot for this information) before and after clicking Replace Me in Listing 17-10.

Listing 17-10: **Invoking the location.replace() Method**

```
<HTML>
<HEAD>
<TITLE>location.replace() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
function doReplace() {
    location.replace("lst17-01.htm")
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myForm">
<INPUT TYPE="button" VALUE="Replace Me" onClick="doReplace()">
</FORM>
</BODY>
</HTML>
```

History Object

Properties

length

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The simple function in Listing 17-11 displays one of two alert messages based on the number of items in the browser's history.

Listing 17-11: **A Browser History Count**

```
<HTML>
<HEAD>
<TITLE>History Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
```

```
function showCount() {
    var histCount = window.history.length
    if (histCount > 5) {
        alert("My, my, you\'ve been busy. You have visited " + histCount +
        " pages so far.")
    } else {
        alert("You have been to " + histCount + " Web pages this session.")
    }
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button" NAME="activity" VALUE="My Activity" onClick="showCount()">
</FORM>
</BODY>
</HTML>
```

Methods

back()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listings 17-12 and 17-13 provide a little workshop in which you can test the behavior of a variety of backward and forward navigation in different browsers. The frameset appears in Figure 3-3. Some features work only in NN4+.

Listing 17-12: **Navigation Lab Frameset**

```
<HTML>
<HEAD>
<TITLE>Back and Forward</TITLE>
</HEAD>
<FRAMESET COLS="45%,55%">
    <FRAME NAME="controller" SRC="lst17-13.htm">
    <FRAME NAME="display" SRC="lst17-01.htm">
</FRAMESET>
</HTML>
```

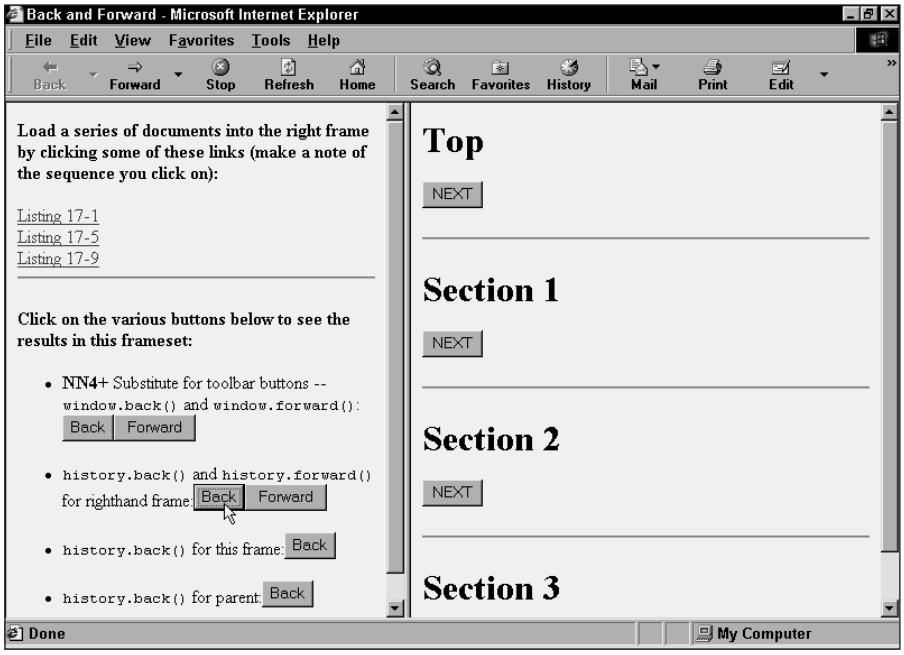


Figure 3-3: Experiment with back and forward behaviors in different browsers

The top portion of Listing 17-13 contains simple links to other example files from this chapter. A click of any link loads a different document into the right-hand frame to let you build some history inside the frame.

Listing 17-13: **Navigation Lab Control Panel**

```
<HTML>
<HEAD>
<TITLE>Lab Controls</TITLE>
</HEAD>
<BODY>
<B>Load a series of documents into the right frame by clicking some of these
links (make a note of the sequence you click on):</B><P>
<A HREF="lst17-01.htm" TARGET="display">Listing 17-1</A><BR>
<A HREF="lst17-05.htm" TARGET="display">Listing 17-5</A><BR>
<A HREF="lst17-09.htm" TARGET="display">Listing 17-9</A><BR>
<HR>
<FORM NAME="input">
<B>Click on the various buttons below to see the results in this
frameset:</B><P>
<UL>
<LI><B>NN4+</B> Substitute for toolbar buttons -- <TT>window.back()</TT> and
<TT>window.forward()</TT>:<INPUT TYPE="button" VALUE="Back"
onClick="window.back()"><INPUT TYPE="button" VALUE="Forward"
onClick="window.forward()"><P>
```

```
<LI><TT> history.back()</TT> and <TT>history.forward()</TT> for righthand frame:
<INPUT TYPE="button" VALUE="Back" onClick="parent.display.history.back()"><INPUT
TYPE="button" VALUE="Forward" onClick="parent.display.history.forward()"><P>

<LI><TT>history.back()</TT> for this frame:<INPUT TYPE="button" VALUE="Back"
onClick="history.back()"><P>

<LI><TT>history.back()</TT> for parent:<INPUT TYPE="button" VALUE="Back"
onClick="parent.history.back()"><P>
</UL>
</FORM>
</BODY>
</HTML>
```

go(*relativeNumber* | "*URLOrTitleSubstring*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Fill in either the number or text field of the page in Listing 17-14 and then click the associated button. The script passes the appropriate kind of data to the `go()` method. Be sure to use negative numbers for visiting a page earlier in the history.

Listing 17-14: **Navigating to an Item in History**

```
<HTML>
<HEAD>
<TITLE>history.go() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doGoNum(form) {
    window.history.go(parseInt(form.histNum.value))
}
function doGoTxt(form) {
    window.history.go(form.histWord.value)
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<B>Calling the history.go() method:</B>
<HR>
```

Continued

Listing 17-14 *(continued)*

```
Enter a number (+/-):<INPUT TYPE="text" NAME="histNum" SIZE=3 VALUE="0">
<INPUT TYPE="button" VALUE="Go to Offset" onClick="doGoNum(this.form)"><P>
Enter a word in a title:<INPUT TYPE="text" NAME="histWord">
<INPUT TYPE="button" VALUE="Go to Match" onClick="doGoTxt(this.form)">
</FORM>
</BODY>
</HTML>
```

✦ ✦ ✦

CHAPTER 4

The Document and Body Objects (Chapter 18)

In This Chapter

Accessing arrays of objects contained by the `document` object

Writing new document content to a window or frame

Managing BODY element scrolling in IE

To include coverage of the `document` object and BODY element object in the same chapter is logical, provided you don't fall into a conceptual trap that has been set during the evolution of document object models. The `document` object has been with us since the beginning. Even though it is an abstract object (that is to say, the object exists simply by virtue of a page loading into the browser, rather than associated with any HTML tag), a number of its properties reflect attributes that are defined in a page's `<BODY>` tag. For instance, the properties for link colors and background images, whose behaviors are set in BODY element attributes, have been exposed via the document object since the earliest days.

In more modern object models (IE4+ and W3C DOM), the BODY element is its own object. The `document` object strengthens its role as a "super-container" of all the HTML element objects in the page. Thus, the BODY element object is a child element of the root `document` object (see Chapter 14 of the *JavaScript Bible* for more details). But now that the BODY element object can expose its own attributes as properties, the `document` object no longer needs to play that role, except for the sake of backward compatibility with scripts written for older browsers. Instead, the `document` object assumes an even greater role, especially in the W3C DOM, by providing critical properties and methods of a global nature for the entire document.

It is clear, of course, that the BODY element has an important role to play. Both the IE4+ and W3C DOMs expose the `document.body` property, which returns a reference to the BODY element of the current document. The IE4+ DOM, however, bestows even more importance to the BODY element, by forcing it to be the frame of reference for how much a document's content scrolls inside a window or frame. All other DOMs put that control into the hands of the window (that is, scrolling the window rather than the BODY element inside the window).

Examples Highlights

- Observe in Listing 18-1 how (backward-compatible) `document` object properties for various colors (`alinkColor` and the like) impact the look of the page. It may be even more important to experience the lack of dynamic control that these properties provide in a variety of browsers.
- See how IE4+/Windows exposes date information about the document in Listing 18-4.
- Listings 18-11 and 18-12 provide a workshop to let you test how well your target browsers support the `document.referrer` property. You may need to put them on your server for the real test. Unfortunately, IE/Windows doesn't always provide the desired information.
- If you script for W3C-DOM compatibility, be sure to grasp the `document.getElementById()` and `document.getElementsByName()` methods with the help of the example steps provided.
- The `document.write()` method is one of the most important ones in the vocabulary. Listings 18-16 through 18-18 demonstrate its power.
- See examples for `document.body.scrollLeft` and `document.body.doScroll()` to control document scrolling in IE, and the `onScroll` event handler example (Listing 18-21) to see how to keep a page scrolled at a fixed position.

Document Object

Properties

activeElement

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) with IE4+ to experiment with the `activeElement` property. Type the following statement into the top text box:

```
document.activeElement.value
```

After you press the Enter key, the Results box shows the value of the text box you just typed into (the very same expression you just typed). But if you then click the Evaluate button, you will see the `value` property of that button object appear in the Results box.

alinkColor
bgColor
fgColor
linkColor
vlinkColor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

I select some color values at random to plug into three settings of the ugly colors group for Listing 18-1. The smaller window displays a dummy button so that you can see how its display contrasts with color settings. Notice that the script sets the colors of the smaller window by rewriting the entire window's HTML code. After changing colors, the script displays the color values in the original window's textarea. Even though some colors are set with the color constant values, properties come back in the hexadecimal triplet values. You can experiment to your heart's content by changing color values in the listing. Every time you change the values in the script, save the HTML file and reload it in the browser.

Listing 18-1: **Color Sampler**

```
<HTML>
<HEAD>
<TITLE>Color Me</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function defaultColors() {
    return "BGCOLOR='#c0c0c0' VLINK='#551a8b' LINK='#0000ff'"
}

function uglyColors() {
    return "BGCOLOR='yellow' VLINK='pink' LINK='lawngreen'"
}
```

Continued

Listing 18-1 *(continued)*

```
function showColorValues() {
    var result = ""
    result += "bgColor: " + newWindow.document.bgColor + "\n"
    result += "vlinkColor: " + newWindow.document.vlinkColor + "\n"
    result += "linkColor: " + newWindow.document.linkColor + "\n"
    document.forms[0].results.value = result
}
// dynamically writes contents of another window
function drawPage(colorStyle) {
    var thePage = ""
    thePage += "<HTML><HEAD><TITLE>Color Sampler</TITLE></HEAD><BODY "
    if (colorStyle == "default") {
        thePage += defaultColors()
    } else {
        thePage += uglyColors()
    }
    thePage += ">Just so you can see the variety of items and color, <A "
    thePage += "HREF='http://www.nowhere.com'>here\'s a link</A>, and " +
               "<A HREF='http://home.netscape.com'> here is another link </A> " +
               "you can use on-line to visit and see how its color differs " +
               "from the standard link."
    thePage += "<FORM>"
    thePage += "<INPUT TYPE='button' NAME='sample' VALUE='Just a Button'>"
    thePage += "</FORM></BODY></HTML>"
    newWindow.document.write(thePage)
    newWindow.document.close()
    showColorValues()
}
// the following works properly only in Windows Navigator
function setColors(colorStyle) {
    if (colorStyle == "default") {
        document.bgColor = "#c0c0c0"
    } else {
        document.bgColor = "yellow"
    }
}
var newWindow = window.open("","","height=150,width=300")
</SCRIPT>
</HEAD>

<BODY>
Try the two color schemes on the document in the small window.
<FORM>
<INPUT TYPE="button" NAME="default" VALUE='Default Colors'
    onClick="drawPage('default')">
<INPUT TYPE="button" NAME="weird" VALUE="Ugly Colors"
    onClick="drawPage('ugly')"><P>
<TEXTAREA NAME="results" ROWS=3 COLS=20></TEXTAREA><P><HR>
These buttons change the current document, but not correctly on all platforms<P>
```

```
<INPUT TYPE="button" NAME="default" VALUE='Default Colors'
    onClick="setColors('default')">
<INPUT TYPE="button" NAME="weird" VALUE="Ugly Colors"
    onClick="setColors('ugly')"><P>
</FORM>
<SCRIPT LANGUAGE="JavaScript">
drawPage("default")
</SCRIPT>
</BODY>
</HTML>
```

To satisfy the curiosity of those who want to change the color of a loaded document on the fly, the preceding example includes a pair of buttons that set the color properties of the current document. If you're running browsers and versions capable of this power (see Table 18-1), everything will look fine; but in other platforms or earlier versions, you may lose the buttons and other document content behind the color. You can still click and activate these items, but the color obscures them. Unless you know for sure that users of your Web page use only browsers and clients empowered for background color changes, do not change colors by setting properties of an existing document.

If you are using Internet Explorer 3 for the Macintosh, you will experience some difficulties with Listing 18-1. The script in the main document loses its connection with the subwindow; it does not redraw the second window with other colors. You can, however, change the colors in the main document. The significant flicker you may experience is related to the way the Mac version redraws content after changing colors.

anchors

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

In Listing 18-2, I append an extra script to Listing 17-1 (in Chapter 3 of this book) to demonstrate how to extract the number of anchors in the document. The document dynamically writes the number of anchors found in the document. You will not likely ever need to reveal such information to users of your page, and the `document.anchors` property is not one that you will call frequently. The object model defines it automatically as a document property while defining actual anchor objects.

Listing 18-2: **Reading the Number of Anchors**

```
<HTML>
<HEAD>
<TITLE>document.anchors Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function goNextAnchor(where) {
    window.location.hash = where
}
</SCRIPT>
</HEAD>

<BODY>

<A NAME="start"><H1>Top</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec1')">
</FORM>
<HR>

<A NAME="sec1"><H1>Section 1</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec2')">
</FORM>
<HR>

<A NAME="sec2"><H1>Section 2</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="NEXT" onClick="goNextAnchor('sec3')">
</FORM>
<HR>

<A NAME="sec3"><H1>Section 3</H1></A>
<FORM>
<INPUT TYPE="button" NAME="next" VALUE="BACK TO TOP"
onClick="goNextAnchor('start')">
</FORM>
<HR><P>
<SCRIPT LANGUAGE="JavaScript">
document.write("<I>There are " + document.anchors.length +
" anchors defined for this document</I>")
</SCRIPT>
</BODY>
</HTML>
```

applets

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

The `document.applets` property is defined automatically as the browser builds the object model for a document that contains applet objects. You will rarely access this property, except to determine how many applet objects a document has.

bgColor

See `alinkColor`.

body

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to examine properties of the BODY element object. First, to prove that the `document.body` is the same as the element object that comes back from longer references, enter the following statement into the top text box with either IE5 or NN6:

```
document.body == document.getElementsByTagName("BODY")[0]
```

Next, check out the BODY object's property listings later in this chapter and enter the listings into the top text box to review their results. For example:

```
document.body.bgColor
document.body.tagName
```

charset

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `charset` property. To see the default setting applied to the page, enter the following statement into the top text box:

```
document.charset
```

If you are running IE5+ for Windows 98 and you enter the following statement, the browser will apply a different character set to the page:

```
document.charset = "iso-8859-2"
```

If your version of Windows does not have that character set installed in the system, the browser may ask permission to download and install the character set.

characterSet

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `characterSet` property in NN6. To see the default setting applied to the page, enter the following statement into the top text box:

```
document.characterSet
```

cookie

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Experiment with the last group of statements in Listing 18-3 to create, retrieve, and delete cookies. You can also experiment with The Evaluator by assigning a name/value pair string to `document.cookie`, and then examining the value of the `cookie` property.

defaultCharset

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `defaultCharset` property. To see the default setting applied to the page, enter the following statement into the top text box:

```
document.defaultCharset
```

documentElement

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to examine the behavior of the `documentElement` property. In IE5+ or NN6, enter the following statement into the top text field:

```
document.documentElement.tagName
```

The result is `HTML`, as expected.

expando

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `document.expando` property in IE4+. Begin by proving that the `document` object can normally accept custom properties. Type the following statement into the top text field:

```
document.spooky = "Boo!"
```

This property is now set and stays that way until the page is either reloaded or unloaded.

Now freeze the `document` object's properties with the following statement:

```
document.expando = false
```

If you try to add a new property, such as the following, you receive an error:

```
document.happy = "tra la"
```

Interestingly, even though `document.expando` is turned off, the first custom property is still accessible and modifiable.

fgColor

See `alinkColor`.

fileCreatedDate
fileModifiedDate
fileSize

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 18-4 dynamically generates several pieces of content relating to the creation and modification dates of the file, as well as its size. More importantly, the listing demonstrates how to turn a value returned by the file date properties into a genuine date object that can be used for date calculations. In the case of Listing 18-4, the calculation is the number of full days between the creation date and the day someone views the file. Notice that the dynamically generated content is added very simply via the `innerText` properties of carefully-located SPAN elements in the body content.

Listing 18-4: **Viewing File Dates**

```
<HTML>
<HEAD>
<TITLE>fileCreatedDate and fileModifiedDate Properties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function fillInBlanks() {
    var created = document.fileCreatedDate
    var modified = document.fileModifiedDate
    document.all.created.innerText = created
    document.all.modified.innerText = modified
    var createdDate = new Date(created).getTime()
    var today = new Date().getTime()
    var diff = Math.floor((today - createdDate) / (1000*60*60*24))
    document.all.diff.innerText = diff
    document.all.size.innerText = document.fileSize
}
</SCRIPT>
</HEAD>

<BODY onLoad="fillInBlanks()">
<H1>fileCreatedDate and fileModifiedDate Properties</H1>
<HR>
```

```
<P>This file (<SPAN ID="size"> </SPAN> bytes) was created
on <SPAN ID="created"> </SPAN> and most
recently modified on <SPAN ID="modified"> </SPAN>.</P>
<P>It has been <SPAN ID="diff"> </SPAN> days since this file was
created.</P>
</BODY>
</HTML>
```

forms

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The document in Listing 18-5 is set up to display an alert dialog box that simulates navigation to a particular music site, based on the checked status of the "bluish" check box. The user input here is divided into two forms: one form with the check box and the other form with the button that does the navigation. A block of copy fills the space in between. Clicking the bottom button (in the second form) triggers the function that fetches the checked property of the "bluish" checkbox by using the `document.forms[i]` array as part of the address.

Listing 18-5: **Using the document.forms Property**

```
<HTML>
<HEAD>
<TITLE>document.forms example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function goMusic() {
    if (document.forms[0].bluish.checked) {
        alert("Now going to the Blues music area...")
    } else {
        alert("Now going to Rock music area...")
    }
}
</SCRIPT>
</HEAD>

<BODY>
<FORM NAME="theBlues">
<INPUT TYPE="checkbox" NAME="bluish">Check here if you've got the blues.
</FORM>
<HR>
```

Continued

Listing 18-5 *(continued)*

```
M<BR>
o<BR>
r<BR>
e<BR>
<BR>
C<BR>
o<BR>
p<BR>
y<BR>
<HR>
<FORM NAME="visit">
<INPUT TYPE="button" VALUE="Visit music site" onClick="goMusic()">
</FORM>
</BODY>
</HTML>
```

frames

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listings 16-7 and 16-8 (in Chapter 2 of this book) for examples of using the `frames` property with window objects. The listings work with IE4+ if you swap references to the `window` with `document`.

height
width

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to examine the `height` and `width` properties of that document. Enter the following statement into the top text box and click the Evaluate button:

```
"height=" + document.height + "; width=" + document.width
```

Resize the window so that you see both vertical and horizontal scrollbars in the browser window and click the Evaluate button again. If either or both numbers get smaller, the values in the Results box are the exact size of the space occupied by the document. But if you expand the window to well beyond where the scrollbars are needed, the values extend to the number of pixels in each dimension of the window's content region.

images

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓	(✓)		✓	✓	✓

Example

The `document.images` property is defined automatically as the browser builds the object model for a document that contains image objects. See the discussion about the `Image` object in Chapter 22 of the *JavaScript Bible* for reference examples.

implementation

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `document.implementation.hasFeature()` method in NN6. Enter the following statements one at a time into the top text field and examine the results:

```
document.implementation.hasFeature("HTML","1.0")
document.implementation.hasFeature("HTML","2.0")
document.implementation.hasFeature("HTML","3.0")
document.implementation.hasFeature("CSS","2.0")
document.implementation.hasFeature("CSS2","2.0")
```

Feel free to try other values.

lastModified

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Experiment with the `document.lastModified` property with Listing 18-6. But also be prepared for inaccurate readings if the file is located on some servers or local hard disks.

Listing 18-6: document.lastModified Property in Another Format

```
<HTML>
<HEAD>
<TITLE>Time Stamper</TITLE>
</HEAD>
<BODY>
<CENTER> <H1>GiantCo Home Page</H1></CENTER>
<SCRIPT LANGUAGE="JavaScript">
update = new Date(document.lastModified)
theMonth = update.getMonth() + 1
theDate = update.getDate()
theYear = update.getFullYear()
document.writeln("<I>Last updated:" + theMonth + "/" + theDate + "/" + theYear +
"</I>")
</SCRIPT>
<HR>
</BODY>
</HTML>
```

As noted at great length in the `Date` object discussion in Chapter 36 of the *JavaScript Bible*, you should be aware that date formats vary greatly from country to country. Some of these formats use a different order for date elements. When you hard-code a date format, it may take a form that is unfamiliar to other users of your page.

layers

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

Listing 18-7 demonstrates only for NN4 how to use the `document.layers` property to crawl through the entire set of nested layers in a document. Using reflexive calls to the `crawlLayers()` function, the script builds an indented list of layers in

the same hierarchy as the objects themselves and displays the results in an alert dialog box. After you load this document (the script is triggered by the `onLoad` event handler), compare the alert dialog box contents against the structure of `<LAYER>` tags in the document.

Listing 18-7: **A Navigator 4 Layer Crawler**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript1.2">
var output = ""
function crawlLayers(layerArray, indent) {
    for (var i = 0; i < layerArray.length; i++) {
        output += indent + layerArray[i].name + "\n"
        if (layerArray[i].document.layers.length) {
            var newLayerArray = layerArray[i].document.layers
            crawlLayers(newLayerArray, indent + "  ")
        }
    }
    return output
}
function revealLayers() {
    alert(crawlLayers(document.layers, ""))
}
</SCRIPT>
</HEAD>
<BODY onLoad="revealLayers()">
<LAYER NAME="Europe">
    <LAYER NAME="Germany"></LAYER>
    <LAYER NAME="Netherlands">
        <LAYER NAME="Amsterdam"></LAYER>
        <LAYER NAME="Rotterdam"></LAYER>
    </LAYER>
    <LAYER NAME="France"></LAYER>
</LAYER>
<LAYER NAME="Africa">
    <LAYER NAME="South Africa"></LAYER>
    <LAYER NAME="Ivory Coast"></LAYER>
</LAYER>
</BODY>
</HTML>
```

linkColor

See `alinkColor`.

links

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The `document.links` property is defined automatically as the browser builds the object model for a document that contains link objects. You rarely access this property, except to determine the number of link objects in the document.

location
URL

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	(✓)	✓	✓	✓	(✓)	(✓)	✓	✓	✓

Example

HTML documents in Listing 18-8 through 18-10 create a test lab that enables you to experiment with viewing the `document.URL` property for different windows and frames in a multiframe environment. Results are displayed in a table, with an additional listing of the `document.title` property to help you identify documents being referred to. The same security restrictions that apply to retrieving `window.location` object properties also apply to retrieving the `document.URL` property from another window or frame.

Listing 18-8: Frameset for document.URL Property Reader

```
<HTML>
<HEAD>
<TITLE>document.URL Reader</TITLE>
</HEAD>
<FRAMESET ROWS="60%,40%">
    <FRAME NAME="Frame1" SRC="lst18-10.htm">
    <FRAME NAME="Frame2" SRC="lst18-09.htm">
</FRAMESET>
</HTML>
```

Listing 18-9: **document.URL Property Reader**

```
<HTML>
<HEAD>
<TITLE>URL Property Reader</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
function fillTopFrame() {
    newURL=prompt("Enter the URL of a document to show in the top frame:","")
    if (newURL != null && newURL != "") {
        top.frames[0].location = newURL
    }
}

function showLoc(form,item) {
    var windName = item.value
    var theRef = windName + ".document"
    form.dLoc.value = unescape(eval(theRef + ".URL"))
    form.dTitle.value = unescape(eval(theRef + ".title"))
}
</SCRIPT>
</HEAD>

<BODY>
Click the "Open URL" button to enter the location of an HTML document to display
in the upper frame of this window.
<FORM>
<INPUT TYPE="button" NAME="opener" VALUE="Open URL..." onClick="fillTopFrame()">
</FORM>
<HR>
<FORM>
Select a window or frame to view each document property values.<P>
<INPUT TYPE="radio" NAME="whichFrame" VALUE="parent"
onClick="showLoc(this.form,this)">Parent window
<INPUT TYPE="radio" NAME="whichFrame" VALUE="top.frames[0]"
onClick="showLoc(this.form,this)">Upper frame
<INPUT TYPE="radio" NAME="whichFrame" VALUE="top.frames[1]"
onClick="showLoc(this.form,this)">This frame<P>
<TABLE BORDER=2>
<TR><TD ALIGN=RIGHT>document.URL:</TD>
<TD><TEXTAREA NAME="dLoc" ROWS=3 COLS=30 WRAP="soft"></TEXTAREA></TD></TR>

<TR><TD ALIGN=RIGHT>document.title:</TD>
<TD><TEXTAREA NAME="dTitle" ROWS=3 COLS=30 WRAP="soft"></TEXTAREA></TD></TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Listing 18-10: **Placeholder for Listing 18-8**

```
<HTML>
<HEAD>
<TITLE>Opening Placeholder</TITLE>
</HEAD>
<BODY>
Initial place holder. Experiment with other URLs for this frame (see below).
</BODY>
</HTML>
```

parentWindow

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

To prove the `parentWindow` property points to the document's window, you can enter the following statement into the top text field of The Evaluator (Chapter 13 in the *JavaScript Bible*):

```
document.parentWindow == self
```

This expression evaluates to `true` only if both references are of the same object.

protocol

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

If you use The Evaluator (Chapter 13 in the *JavaScript Bible*) to test the `document.protocol` property, you will find that it displays `File Protocol` in the results because you are accessing the listing from a local hard disk or CD-ROM.

referrer

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

This demonstration requires two documents (and for IE, you'll also need to access the documents from a Web server). The first document, in Listing 18-11, simply contains one line of text as a link to the second document. In the second document (Listing 18-12), a script verifies the document from which the user came via a link. If the script knows about that link, it displays a message relevant to the experience the user had at the first document. Also try opening Listing 18-12 in a new browser window from the Open File command in the File menu to see how the script won't recognize the referrer.

Listing 18-11: A Source Document

```
<HTML>
<HEAD>
<TITLE>document.referrer Property 1</TITLE>
</HEAD>

<BODY>
<H1><A HREF="lst18-12.htm">Visit my sister document</A>
</BODY>
</HTML>
```

Listing 18-12: Checking document.referrer

```
<HTML>
<HEAD>
<TITLE>document.referrer Property 2</TITLE>
</HEAD>

<BODY><H1>
<SCRIPT LANGUAGE="JavaScript">
if(document.referrer.length > 0 &&
document.referrer.indexOf("18-11.htm") != -1){
    document.write("How is my brother document?")
} else {
    document.write("Hello, and thank you for stopping by.")
}
</SCRIPT>
</H1></BODY>
</HTML>
```

scripts

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

You can experiment with the `document.scripts` array in The Evaluator (Chapter 13 in the *JavaScript Bible*). For example, you can see that only one SCRIPT element object is in The Evaluator page if you enter the following statement into the top text field:

```
document.scripts.length
```

If you want to view all of the properties of that lone SCRIPT element object, enter the following statement into the bottom text field:

```
document.scripts[0]
```

Among the properties are both `innerText` and `text`. If you assign an empty string to either property, the scripts are wiped out from the object model, but not from the browser. The scripts disappear because after the scripts loaded, they were cached outside of the object model. Therefore, if you enter the following statement into the top field:

```
document.scripts[0].text = ""
```

the script contents are gone from the object model, yet subsequent clicks of the Evaluate and List Properties buttons (which invoke functions of the SCRIPT element object) still work.

selection

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listings 15-30 and 15-39 in Chapter 1 of this book to see the `document.selection` property in action for script-controlled copying and pasting (IE/Windows only).

URL

See `location`.

vlinkColor

See alinkColor.

width

See height.

Methods

captureEvents(*eventTypeList*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

See the example for the NN4 window.captureEvents() method (Listing 16-21 from Chapter 2 of this book) to see how to capture events on their way to other objects. In that example, you can substitute the document reference for the window reference to see how the document version of the method works just like the window version. If you understand the mechanism for windows, you understand it for documents. The same is true for the other NN4 event methods.

close()

NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5	
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Before you experiment with the document.close() method, be sure you understand the document.write() method described later in this chapter. After that, make a separate set of the three documents for that method's example (Listings 18-16 through 18-18 in a different directory or folder). In the takePulse() function listing, comment out the document.close() statement, as shown here:

```
msg += "<P>Make it a great day!</BODY></HTML>"
parent.frames[1].document.write(msg)
//parent.frames[1].document.close()
```

Now try the pages on your browser. You see that each click of the upper button appends text to the bottom frame, without first removing the previous text. The reason is that the previous layout stream was never closed. The document thinks that you're still writing to it. Also, without properly closing the stream, the last line of text may not appear in the most recently written batch.

createAttribute("*attributeName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Unfortunately, the `setAttributeNode()` method in NN6 does not yet work with attributes generated by the `createAttribute()` method. This will be fixed eventually, and you can experiment adding attributes to sample elements in The Evaluator. In the meantime, you can still create an attribute and inspect its properties. Enter the following text into the top text box:

```
a = document.createAttribute("author")
```

Now enter `a` into the bottom text box to inspect the properties of an `Attr` object.

createElement("*tagName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Chapter 15 of the *JavaScript Bible* contains numerous examples of the `document.createElement()` method in concert with methods that add or replace content to a document. See Listings 15-10, 15-21, 15-22,15-28, 15-29, and 15-31 in Chapter 1 of this book.

createEventObject([*eventObject*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility									✓

Example

See the discussion of the `fireEvent()` method in Chapter 15 of the *JavaScript Bible* for an example of the sequence to follow when creating an event to fire on an element.

`createStyleSheet(["URL"[, index]])`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 18-13 demonstrates adding an internal and external style sheet to a document. For the internal addition, the `addStyle1()` function invokes `document.createStyleSheet()` and adds a rule governing the P elements of the page (not available for IE5/Mac). In the `addStyle2()` function, an external file is loaded. That file contains the following two style rules:

```
H2 {font-size:20pt; color:blue}
P  {color:blue}
```

Notice that by specifying a position of zero for the imported style sheet, the addition of the internal style sheet always comes afterward in `styleSheet` object sequence. Thus, except when you deploy only the external style sheet, the red text color of the P elements overrides the blue color of the external style sheet. If you remove the second parameter of the `createStyleSheet()` method in `addStyle2()`, the external style sheet is appended to the end of the list. If it is the last style sheet to be added, the blue color prevails. Repeatedly clicking the buttons in this example continues to add the style sheets to the document.

Listing 18-13: **Using document.createStyleSheet()**

```
<HTML>
<HEAD>
<TITLE>document.createStyleSheet() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function addStyle1() {
    var newStyle = document.createStyleSheet()
    newStyle.addRule("P", "font-size:16pt; color:red")
}

function addStyle2() {
    var newStyle = document.createStyleSheet("lst18-13.css",0)
}
```

Continued

Listing 18-13 *(continued)*

```
</SCRIPT>
</HEAD>

<BODY>
<H1>document.createStyleSheet() Method</H1>
<HR>
<FORM>
<INPUT TYPE="button" VALUE="Add Internal" onClick="addStyle1()"> 
<INPUT TYPE="button" VALUE="Add External" onClick="addStyle2()">
</FORM>
<H2>Section 1</H2>
<P>Lorem ipsum dolor sit amet, consectetaur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim adminim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.</P>
<H2>Section 2</H2>
<P>Duis aute irure dolor in reprehenderit involuptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deseruntmollit
anim id est laborum.</P>
</BODY>
</HTML>
```

createTextNode("*text*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

While Chapter 14 and 15 of the *JavaScript Bible* (Listing 15-21 in Chapter 1, for instance) provide numerous examples of the `createTextNode()` method at work, using The Evaluator (Chapter 13 in the *JavaScript Bible*) is instructive to see just what the method generates in IE5+ and NN6. You can use one of the built-in global variables of The Evaluator to hold a reference to a newly generated text node by entering the following statement into the top text field:

```
a = document.createTextNode("Hello")
```

The Results box shows that an object was created. Now, look at the properties of the object by typing `a` into the bottom text field. The precise listings of properties varies between IE5+ and NN6, but the W3C DOM properties that they share in common indicate that the object is a node type 3 with a node name of `#text`. No parents, children, or siblings exist yet because the object created here is not part of the document hierarchy tree until it is explicitly added to the document.

To see how insertion works, enter the following statement into the top text field to append the text node to the `myP` paragraph:

```
document.getElementById("myP").appendChild(a)
```

The word "Hello" appears at the end of the simple paragraph lower on the page. Now you can modify the text of that node either via the reference from the point of view of the containing P element or via the global variable reference for the newly created node:

```
document.getElementById("myP").lastChild.nodeValue = "Howdy"
```

or

```
a.nodeValue = "Howdy"
```

elementFromPoint(*x*, *y*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 18-14 is a document that contains many different types of elements, each of which has an ID attribute assigned to it. The `onMouseOver` event handler for the `document` object invokes a function that finds out which element the cursor is over when the event fires. Notice that the event coordinates are `event.clientX` and `event.clientY`, which use the same coordinate plane as the page for their point of reference. As you roll the mouse over every element, its ID appears on the page. In Figure 4-1, the pointer is inside a table cell, whose ID appears in bold at the end of the first paragraph. Some elements, such as BR and TR, occupy no space in the document, so you cannot get their IDs to appear. On a typical browser screen size, a positioned element rests atop one of the paragraph elements so that you can see how the `elementFromPoint()` method handles overlapping elements. If you scroll the page, the coordinates for the event and the page's elements stay in sync.

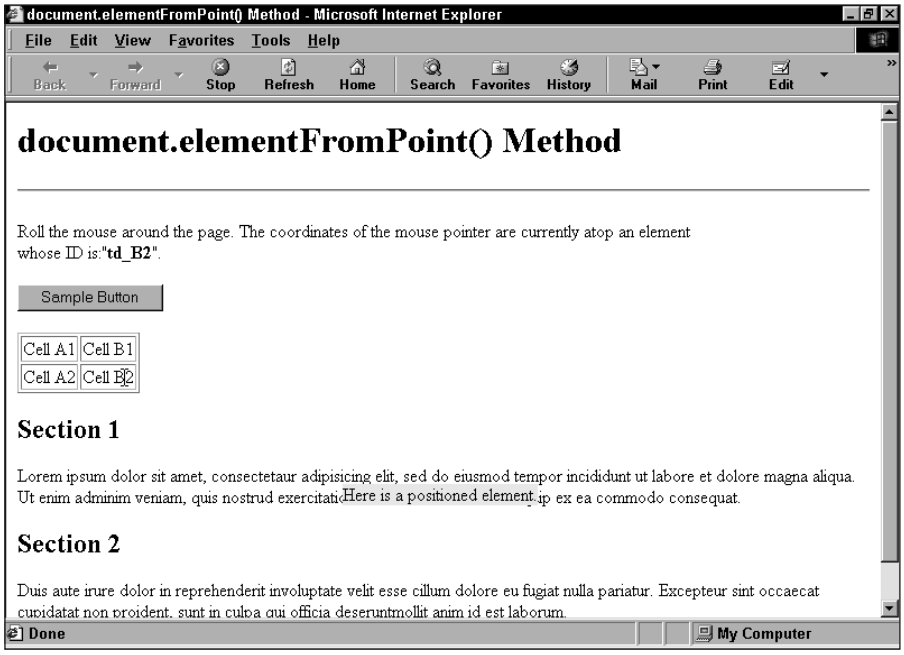


Figure 4-1: Revealing the object located at an event screen position

Listing 18-14: **Using the elementFromPoint() Method**

```
<HTML>
<HEAD>
<TITLE>document.elementFromPoint() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function showElemUnderneath() {
    var elem = document.elementFromPoint(event.clientX, event.clientY)
    document.all.mySpan.innerText = elem.id
}
document.onmouseover = showElemUnderneath
</SCRIPT>
</HEAD>

<BODY ID="myBody">
<H1 ID="header">document.elementFromPoint() Method</H1>
<HR ID="myHR">
<P ID="instructions">Roll the mouse around the page. The coordinates
of the mouse pointer are currently atop an element<BR ID="myBR">whose ID
is:"<SPAN ID="mySpan" STYLE="font-weight:bold"></SPAN>".</P>
<FORM ID="myForm">
<INPUT ID="myButton" TYPE="button" VALUE="Sample Button" onClick=""> 
</FORM>
<TABLE BORDER=1 ID="myTable">
```

```
<TR ID="tr1">
    <TD ID="td_A1">Cell A1</TD>
    <TD ID="td_B1">Cell B1</TD>
</TR>
<TR ID="tr2">
    <TD ID="td_A2">Cell A2</TD>
    <TD ID="td_B2">Cell B2</TD>
</TR>
</TABLE>
<H2 ID="sec1">Section 1</H2>
<P ID="p1">Lorem ipsum dolor sit amet, consectetaur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim adminim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.</P>
<H2 ID="sec2">Section 2</H2>
<P ID="p2">Duis aute irure dolor in reprehenderit involuptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deseruntmollit
anim id est laborum.</P>
<DIV ID="myDIV" STYLE="position:absolute; top:340; left:300; background-
color:yellow">
Here is a positioned element.</DIV>
</BODY>
</HTML>
```

execCommand("*commandName*"[, *UIFlag*] [, *param*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

You can find many examples of the `execCommand()` method for the `TextRange` object in Chapter 19 of the *JavaScript Bible*. But you can try out the document-specific commands in The Evaluator (Chapter 13 in the *JavaScript Bible*) if you like. Try each of the following statements in the top text box and click the Evaluate button:

```
document.execCommand("Refresh")
document.execCommand("SelectAll")
document.execCommand("Unselect")
```

All methods return `true` in the Results box.

Because any way you can evaluate a statement in The Evaluator forces a body selection to become deselected before the evaluation takes place, you can't experiment this way with the selection-oriented commands.

getElementById("*elementID*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

You can find many examples of this method in use throughout this book, but you can take a closer look at how it works by experimenting in The Evaluator (Chapter 13 in the *JavaScript Bible*). A number of elements in The Evaluator have IDs assigned to them, so that you can use the method to inspect the objects and their properties. Enter the following statements into both the top and bottom text fields of The Evaluator. Results from the top field are references to the objects; results from the bottom field are lists of properties for the particular object.

```
document.getElementById("myP")
document.getElementById("myEM")
document.getElementById("myTitle")
document.getElementById("myScript")
```

As you see in the Results field, NN6 is more explicit about the type of HTML element object being referenced in the top text field than IE5. But nevertheless, both browsers are pointing to the same objects.

getElementsByName("*elementName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Use The Evaluator to test out the `getElementsByName()` method. All form elements in the upper part of the page have names associated with them. Enter the following statements into the top text field and observe the results:

```
document.getElementsByName("output")
document.getElementsByName("speed").length
document.getElementsByName("speed")[0].value
```

You can also explore all of the properties of the text field by typing the following expression into the bottom field:

```
document.getElementsByName("speed")[0]
```

getSelection()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

The document in Listing 18-15 provides a cross-browser (but not IE5/Mac) solution to capturing text that a user selects in the page. Selected text is displayed in the textarea. The script uses browser detection and branching to accommodate the diverse ways of recognizing the event and reading the selected text.

Listing 18-15: **Capturing a Text Selection**

```
<HTML>
<HEAD>
<TITLE>Getting Selected Text</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var isNav4 = (navigator.appName == "Netscape"
    && parseInt(navigator.appVersion) == 4)
var isNav4Min = (navigator.appName == "Netscape" &&
    parseInt(navigator.appVersion) >= 4)
var isIE4Min = (navigator.appName.indexOf("Microsoft") != -1 &&
    parseInt(navigator.appVersion) >= 4)
function showSelection() {
    if (isNav4Min) {
        document.forms[0].selectedText.value = document.getSelection()
    } else if (isIE4Min) {
        if (document.selection) {
            document.forms[0].selectedText.value =
                document.selection.createRange().text
            event.cancelBubble = true
        }
    }
}
if (isNav4) {
    document.captureEvents(Event.MOUSEUP)
}
document.onmouseup = showSelection
</SCRIPT>
</HEAD>

<BODY>
<H1>Getting Selected Text</H1>
<HR>
<P>Select some text and see how JavaScript can capture the selection:</P>
<H2>ARTICLE I</H2>
<P>
```

Continued

Listing 18-15 *(continued)*

```
Congress shall make no law respecting an establishment of religion, or
prohibiting the
free exercise thereof; or abridging the freedom of speech, or of the press; or
the right of the people peaceably to assemble, and to petition the government
for a redress of grievances.
</P>
</HR>
<FORM>
<TEXTAREA NAME="selectedText" ROWS=3 COLS=40 WRAP="virtual"></TEXTAREA>
</FORM>
</BODY>
</HTML>
```

open(["*mimeType*"] [, *replace*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

You can see an example of where the `document.open()` method fits in the scheme of dynamically creating content for another frame in the discussion of the `document.write()` method later in this chapter.

queryCommandEnabled("*commandName*")
queryCommandIndterm("*commandName*")
queryCommandCommandState("*commandName*")
queryCommandSupported("*commandName*")
queryCommandText("*commandName*")
queryCommandValue("*commandName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See the examples for these methods covered under the `TextRange` object in Chapter 19 of the *JavaScript Bible*.

```
recalc([allFlag])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

You can see an example of `recalc()` in Listing 15-32 (in Chapter 1 of this book) for the `setExpression()` method. In that example, the dependencies are between the current time and properties of standard element objects.

```
write("string1" [,"string2" ...
[, "stringn"]])
writeln("string1" [,"string2" ...
[, "stringn"]])
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The example in Listings 18-16 through 18-18 demonstrates several important points about using the `document.write()` or `document.writeln()` methods for writing to another frame. First is the fact that you can write any HTML code to a frame, and the browser accepts it as if the source code came from an HTML file somewhere. In the example, I assemble a complete HTML document, including basic HTML tags for completeness.

Listing 18-16: Frameset for document.write() Example

```
<HTML>
<HEAD>
<TITLE>Writin' to the doc</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%">
    <FRAME NAME="Frame1" SRC="lst18-17.htm">
    <FRAME NAME="Frame2" SRC="lst18-18.htm">
</FRAMESET>
</HTML>
```

Listing 18-17: **document.write() Example**

```
<HTML>
<HEAD>
<TITLE>Document Write Controller</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function takePulse(form) {
    var msg = "<HTML><HEAD><TITLE>On The Fly with " + form.yourName.value +
        "</TITLE></HEAD>"
    msg += "<BODY BGCOLOR='salmon'><H1>Good Day " + form.yourName.value +
        "!</H1><HR>"
    for (var i = 0; i < form.how.length; i++) {
        if (form.how[i].checked) {
            msg += form.how[i].value
            break
        }
    }
    msg += "<P>Make it a great day!</BODY></HTML>"
    parent.Frame2.document.write(msg)
    parent.Frame2.document.close()
}
function getTitle() {
    alert("Lower frame document.title is now:" + parent.Frame2.document.title)
}
</SCRIPT>
</HEAD>

<BODY>
Fill in a name, and select how that person feels today. Then click "Write To
Below"
to see the results in the bottom frame.
<FORM>
Enter your first name:<INPUT TYPE="text" NAME="yourName" VALUE="Dave"><P>
How are you today? <INPUT TYPE="radio" NAME="how"
VALUE="I hope that feeling continues forever." CHECKED>Swell
<INPUT TYPE="radio" NAME="how" VALUE="You may be on your way to feeling Swell">
Pretty Good
<INPUT TYPE="radio" NAME="how" VALUE="Things can only get better from here.">
So-So<P>
<INPUT TYPE="button" NAME="enter" VALUE="Write To Below"
    onClick="takePulse(this.form)">
<HR>
<INPUT TYPE="button" NAME="peek" VALUE="Check Lower Frame Title"
    onClick="getTitle()">
</BODY>
</HTML>
```

Listing 18-18: **Placeholder for Listing 18-16**

```
<HTML>
<HEAD>
<TITLE>Placeholder</TITLE>
<BODY>
</BODY>
</HTML>
```

Figure 4-2 shows an example of the frame written by the script.

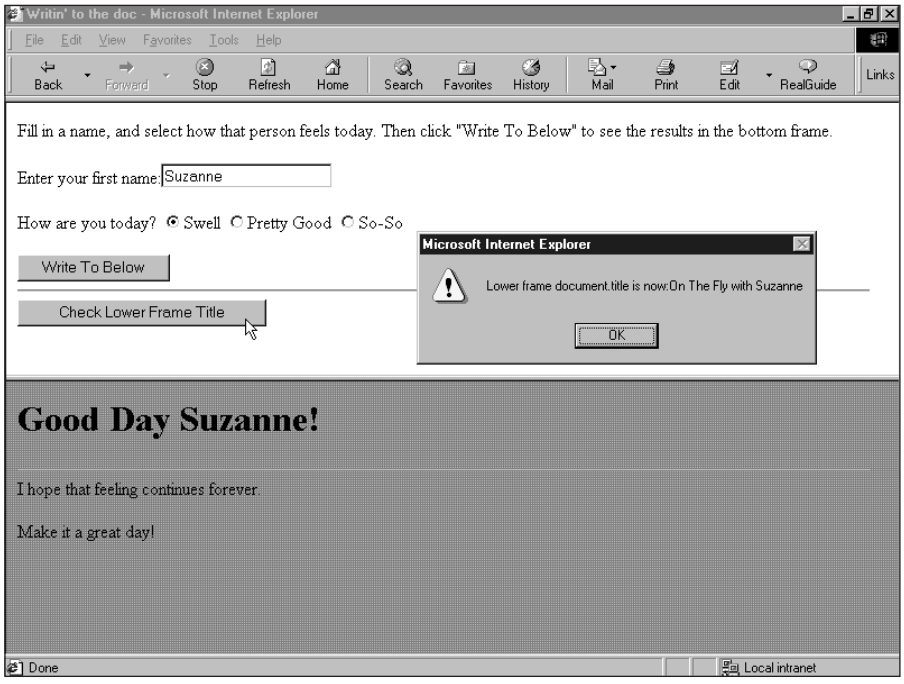


Figure 4-2: Clicking the Write To Below button in the upper frame causes a script to assemble and write HTML for the bottom frame.

A second point to note is that this example customizes the content of the document based on user input. This customization makes the experience of working with your Web page feel far more interactive to the user — yet you're doing it without any CGI programs running on the server.

The third point I want to bring home is that the document created in the separate frame by the `document.write()` method is a genuine `document` object. In this example, for instance, the `<TITLE>` tag of the written document changes if you redraw the lower frame after changing the entry of the name field in the upper frame. If you click the lower button after updating the bottom frame, you see that the `document.title` property has, indeed, changed to reflect the `<TITLE>` tag written to the browser in

the course of displaying the frame's page (except in NN4/Mac, which exhibits a bug for this property in a dynamically written document). The fact that you can artificially create full-fledged JavaScript `document` objects on the fly represents one of the most important powers of serverless CGI scripting (for information delivery to the user) with JavaScript. You have much to take advantage of here if your imagination is up to the task.

Notice that except for NN2, you can easily modify Listing 18-17 to write the results to the same frame as the document containing the field and buttons. Instead of specifying the lower frame

```
parent.frames[1].document.open()
parent.frames[1].document.write(msg)
parent.frames[1].document.close()
```

the code simply can use

```
document.open()
document.write(msg)
document.close()
```

This code would replace the form document with the results and not require any frames in the first place. Because the code assembles all of the content for the new document into one variable value, that data survives the one `document.write()` method.

The frameset document (Listing 18-18) creates a blank frame by loading a blank document (Listing 18-18). An alternative I highly recommend is to have the frame-setting document fill the frame with a blank document of its own creation. See "Blank Frames" in Chapter 16 of the *JavaScript Bible* for further details about this technique for NN3+ and IE3+.

Event Handlers

onStop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 18-19 provides a simple example of an intentional infinitely looping script. In case you load this page into a browser other than IE5, you can click the Halt Counter button to stop the looping. The Halt Counter button and the `onStop` event handler invoke the same function.

Listing 18-19: Scripting the Browser Stop Button

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

```
<TITLE>onStop Event Handler</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var counter = 0
var timerID
function startCounter() {
    document.forms[0].display.value = ++counter
    //clearTimeout(timerID)
    timerID = setTimeout("startCounter()", 10)
}
function haltCounter() {
    clearTimeout(timerID)
    counter = 0
}
document.onstop = haltCounter
</SCRIPT>
</HEAD>

<BODY>
<H1>onStop Event Handler</H1>
<HR>
<P>Click the browser's Stop button (in IE) to stop the script counter.</P>
<FORM>
<P><INPUT TYPE="text" NAME="display"></P>
<INPUT TYPE="button" VALUE="Start Counter" onClick="startCounter()">
<INPUT TYPE="button" VALUE="Halt Counter" onClick="haltCounter()">
</FORM>
</BODY>
</HTML>
```

BODY Element Object

Properties

aLink
bgColor
link
text
vLink

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

You can modify Listing 18-1 for use with IE4+ and NN6+ only by using the new property names instead. Replace all references to the `document` properties with their `document.body` equivalents. For example, the function would be reworked as the following (changes in boldface):

```
function showColorValues() {
    var result = ""
    result += "bgColor: " + newWindow.document.body.bgColor + "\n"
    result += "vLink: " + newWindow.document.body.vLink + "\n"
    result += "link: " + newWindow.document.body.link + "\n"
    document.forms[0].results.value = result
}
```

background

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

If you have a background image file named `images/logoBG.gif`, a script can set the background via the following statement:

```
document.body.background = "images/logoBG.gif"
```

To clear the background image:

```
document.body.background = ""
```

If a background color has been previously set, the color becomes visible after the image disappears.

bgColor

See `aLink`.

bgProperties

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Both of the following statements change the default behavior of background image scrolling in IE4+:

```
document.body.bgProperties = "fixed"
```

or

```
document.body.style.backgroundAttachment = "fixed"
```

The added benefit of using the style sheet version is that it also works in NN6.

bottomMargin
leftMargin
rightMargin
topMargin

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Both of the following statements change the default left margin in IE4+:

```
document.body.leftMargin = 30
```

or

```
document.body.style.marginLeft = 30
```

leftMargin

See `bottomMargin`.

link

See `aLink`.

noWrap

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

To change the word wrapping behavior from the default, the statement is:

```
document.body.noWrap = true
```

rightMargin

See bottomMargin.

scroll

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

To change the scrollbar appearance from the default, the statement is:

```
document.body.scroll = "no"
```

scrollLeft
scrollTop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 18-20 is the IE4+ version of the NN example for pageXOffset and pageYOffset properties (Listing 16-13 in Chapter 2). Everything about these two examples is the same except for the syntax that retrieves the values indicating how much the document is scrolled in a window.

Listing 18-20: Viewing the scrollLeft and scrollTop Properties

```
<HTML>
<HEAD>
<TITLE>Master of all Windows</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function leftFrame() {
    var output = "<HTML><BODY><H3>Body Scroll Values</H3><HR>\n"
    output += "<FORM>body.scrollLeft:<INPUT TYPE='text' NAME='xOffset'
SIZE=4><BR>\n"
    output += "body.scrollTop:<INPUT TYPE='text' NAME='yOffset' SIZE=4><BR>\n"
    output += "</FORM></BODY></HTML>"
    return output
}
```

```
function rightFrame() {
    var output = "<HTML><HEAD><SCRIPT LANGUAGE='JavaScript'>\n"
    output += "function showOffsets() {\n"
    output += "parent.readout.document.forms[0].xOffset.value =  " +
        "document.body.scrollLeft\n"
    output += "parent.readout.document.forms[0].yOffset.value =  " +
        "document.body.scrollTop\n}\n"
    output += "document.onclick = showOffsets\n"
    output += "<\/SCRIPT></HEAD><BODY><H3>Content Page</H3>\n"
    output += "Scroll this frame and click on a table border to view " +
        "page offset values.<BR><HR>\n"
    output += "<TABLE BORDER=5 WIDTH=800>"
    var oneRow = "<TD>Cell 1</TD><TD>Cell 2</TD><TD>Cell 3</TD><TD>Cell 4</TD>" +
        "<TD>Cell 5</TD>"
    for (var i = 1; i <= 30; i++) {
         output += "<TR><TD><B>Row " + i + "</B></TD>" + oneRow + "</TR>"
    }
    output += "</TABLE></BODY></HTML>"
    return output
}
</SCRIPT>
</HEAD>
<FRAMESET COLS="30%,70%">
    <FRAME NAME="readout" SRC="javascript:parent.leftFrame()">
    <FRAME NAME="display" SRC="javascript:parent.rightFrame()">
</FRAMESET>
</HTML>
```

text

See aLink.

topMargin

See bottomMargin.

vLink

See aLink.

Methods

createTextRange()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-8 (in Chapter 5 of this book) for an example of the `createTextRange()` method in action.

doScroll(["*scrollAction*"])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `doScroll()` method in IE5+. Size the browser window so that at least the vertical scrollbar is active (meaning it has a thumb region). Enter the following statement into the top text field and press Enter a few times to simulate clicking the PgDn key:

```
document.body.doScroll()
```

Return to the top of the page and now do the same for scrolling by the increment of the scrollbar down arrow:

```
document.body.doScroll("down")
```

You can also experiment with upward scrolling. Enter the desired statement in the top text field and leave the text cursor in the field. Manually scroll to the bottom of the page and then press Enter to activate the command.

Event Handlers

onAfterPrint
onBeforePrint

See the `onAfterPrint` event handler for the `window` object, Chapter 16 of the *JavaScript Bible*.

onScroll

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 18-21 is a highly artificial demonstration of what can be a useful tool for some page designs. Consider a document that occupies a window or frame, but one that you don't want scrolled, even by accident with one of the newer mouse wheels

that are popular with Wintel PCs. If scrolling of the content would destroy the appearance or value of the content, then you want to make sure that the page always zips back to the top. The `onScroll` event handler in Listing 18-21 does just that. Notice that the event handler is set as a property of the `document.body` object after the page has loaded. While the event handler can also be set as an attribute of the `<BODY>` tag, to assign it as a property requires the page to load first. Until then, the `document.body` object does not yet officially exist in the object model for this page.

Listing 18-21: **Forcing Scrolling to Stay at the Page Top**

```
<HTML>
<HEAD>
<TITLE>onScroll Event Handler</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function zipBack() {
    window.scroll(0,0)
}
function init() {
    document.body.onscroll = zipBack
}
</SCRIPT>
</HEAD>

<BODY onLoad="init()">
<H1>onScroll Event Handler</H1>
<HR>
This page always zips back to the top if you try to scroll it.
<P>
<IFRAME FRAMEBORDER=0 SCROLLING="no" HEIGHT=1000 SRC="bofright.htm"></IFRAME>
</P>
</BODY>
</HTML>
```

✦ ✦ ✦

CHAPTER 5

Body Text Objects (Chapter 19)

The subject of body text objects encompasses both HTML element objects and several abstract DOM objects that make it easier for scripts to manipulate text-oriented body content that may not be contained within its own element tag. While the HTML element objects are easy to grasp, the abstract objects that work with stretches of visible body text have their own vocabularies and peculiarities.

Many HTML element objects in this category may become obsolete when the installed base of browsers capable of supporting Cascading Style Sheets reaches critical mass. CSS adherents would much rather use style sheets for font specifications in place of the old-fashioned `<FONT>` tag. But other elements in this group, such as the header elements (H1, H2, and so on), provide context for content that scripts may find useful for tasks such as creating a table of contents on the fly.

More intriguing is the concept of a text range, which is essentially an object that represents an arbitrary series of text characters within a document. A text ranges can work within an element (or text node) or extend beyond element borders, just as if a user selected a bunch of text that includes portions of what are HTML elements behind the scenes.

Unfortunately for scripters, the vocabulary for text range manipulation is very different for the IE4+/Windows and W3C object models. Moreover, the two objects do not always share the same functionality, making it even more difficult to program cross-browser implementations using text ranges. Be alert to the compatibility ratings for each example before trying out a listing or step-by-step sequence.

In This Chapter

Using the NN `Range` and IE `TextRange` objects

Working with text selections

Scripting search and replace actions

Examples Highlights

- ✦ Many site visitors (this author included) frown on the application of the scrolling MARQUEE element because it tends to distract visitors, rather than convey meaningful information. But if you insist on using it, Listing 19-3 demonstrates how scripts can control numerous behaviors.
- ✦ Listing 19-4 lets you examine how the NN6 (W3C DOM) `Range` object treats boundary points within the node hierarchy of a document.
- ✦ To insert a node into an arbitrary point within another, see Listing 19-5's application of the `Range.insertNode()` method.
- ✦ Walk through the steps for `Range.selectNode()` method to see how to set a range to encompass an entire node or its contents.
- ✦ Run Listing 19-8 to see how NN6 (W3C DOM) provides additional facilities for manipulating text content within a node. The listing also demonstrates `try-catch` error handling.
- ✦ Listing 19-10 shows the IE4+/Windows `TextRange` object's way of comparing range boundaries (the IE version of Listing 19-4).
- ✦ The `TextRange` object provides practical text search facilities, which are demonstrated in Listing 19-11. In the process, several `TextRange` properties and methods get a workout, including the use of bookmarks within a range. A simple undo buffer adds to the user friendliness of the application.

FONT Element Object

Properties

color

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listing 19-1 contains a page that demonstrates changes to the three FONT element object properties: `color`, `face`, and `size`. Along the way, you can see an economical use of the `setAttribute()` method to do the work for all of the property changes. This page loads successfully in all browsers, but the `SELECT` lists make changes to the text only in IE4+ and NN6+.

A P element contains a nested FONT element that encompasses three words whose appearance is controlled by three select lists. Each list controls one of the

three FONT object properties, and their NAME attributes are strategically assigned the names of the properties (as you see in a moment). VALUE attributes for OPTION elements contain strings that are to be assigned to the various properties. Each SELECT element invokes the same setFontAttr() function, passing a reference to itself so that the function can inspect details of the element.

The first task of the setFontAttr() function is to make sure that only browsers capable of treating the FONT element as an object get to the meat of the function. The test for the existence of document.all and the myFONT element blocks all older browsers from changing the font characteristics. As the page loads, the document.all property is set for NN6 by using a variation of the normalization technique described in Chapter 14 of the *JavaScript Bible*.

For suitably equipped browsers, the function next extracts the string from the value property of the SELECT object that was passed to the function. If a selection is made (meaning other than the first, empty one), then the single nested statement uses the setAttribute() method to assign the value to the attribute whose name matches the name of the SELECT element.

An odd bug in IE5/Mac doesn't let the rendered color change when changing the color property. But the setting is valid, as proven by selecting any of the other two property choices.

Listing 19-1: **Controlling FONT Object Properties**

```
<HTML>
<HEAD>
<TITLE>FONT Object Properties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// document.all normalization trick for NN6
if (navigator.appName == "Netscape" && parseInt(navigator.appVersion) >= 5) {
   document.all = document.getElementsByTagName("*")
}

// one function does all!
function setFontAttr(select) {
    if (document.all && document.all.myFONT) {
        var choice = select.options[select.selectedIndex].value
        if (choice) {
            document.all.myFONT.setAttribute(select.name, choice)
        }
    }
}
</SCRIPT>
</HEAD>

<BODY>
<H1>Font Object Properties</H1>
<BR>
```

Continued

Listing 19-1 *(continued)*

```
<P>This may look like a simple sentence, but
<FONT ID="myFONT">THESE THREE WORDS</FONT>
are contained by a FONT element.</P>

<FORM>
Select a text color:
<SELECT NAME="color" onChange="setFontAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="red">Red</OPTION>
    <OPTION VALUE="green">Green</OPTION>
    <OPTION VALUE="blue">Blue</OPTION>
    <OPTION VALUE="#FA8072">Some Hex Triplet Value</OPTION>
</SELECT>
<BR>
Select a font face:
<SELECT NAME="face" onChange="setFontAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="Helvetica">Helvetica</OPTION>
    <OPTION VALUE="Times">Times</OPTION>
    <OPTION VALUE="Comic Sans MS, sans-serif">Comic Sans MS, sans-serif</OPTION>
    <OPTION VALUE="Courier, monospace">Courier, monospace</OPTION>
    <OPTION VALUE="Zapf Dingbats, serif">Zapf Dingbats, serif</OPTION>
</SELECT>
<BR>
Select a font size:
<SELECT NAME="size" onChange="setFontAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="3">3 (Default)</OPTION>
    <OPTION VALUE="+1">Increase Default by 1</OPTION>
    <OPTION VALUE="-1">Decrease Default by 1</OPTION>
    <OPTION VALUE="1">Smallest</OPTION>
    <OPTION VALUE="7">Biggest</OPTION>
</SELECT>
</BODY>
</HTML>
```

face

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See Listing 19-1 for an example of values that can be used to set the `face` property of a FONT element object. While you will notice visible changes to most choices on the page, the font face selections may not change from one choice to another; this all depends on the fonts that are installed on your PC.

`size`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See Listing 19-1 for an example of values that can be used to set the `size` property of a FONT element object. Notice that incrementing or decrementing the `size` property is applied only to the size assigned to the `SIZE` attribute of the element (or the default, if none is specified) and not the current setting adjusted by script.

HR Element Object

Properties

`align`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listing 19-2 contains a page that demonstrates the changes to the five HR element object properties: `align`, `color`, `noShade`, `size`, and `width`. Along the way, you can see an economical use of the `setAttribute()` method to do the work for all of the property changes. This page loads successfully in all browsers, but the SELECT lists make changes to the text only in IE4+ and NN6+ (because they treat the element as an object).

An HR element (whose ID is `myHR`) is displayed with the browser default settings (100% width, centered, and its "magic" color). Each list controls one of the five HR object properties, and their `NAME` attributes are strategically assigned the names of the properties (as you see in a moment). `VALUE` attributes for OPTION elements contain strings that are to be assigned to the various properties. Each SELECT element

invokes the same `setHRAttr()` function, passing a reference to itself so that the function can inspect details of the element. Figure 5-1 shows the page after several choices have modified the HR element.

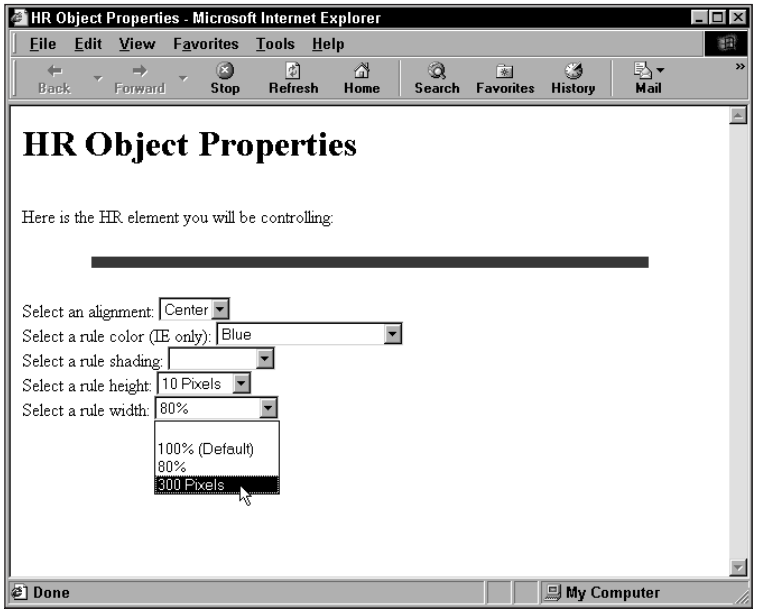


Figure 5-1: Modifying HR element properties

The first task of the `setHRAttr()` function is to make sure that only browsers capable of treating the `HR` element as an object get to the meat of the function. As the page loads, the `document.all` property is set for NN6 using a normalization technique described in Chapter 14 of the *JavaScript Bible*.

For suitably equipped browsers, the function next reads the string from the `value` property of the SELECT object that is passed to the function. If a selection is made (that is, other than the first, empty one), then the single, nested statement uses the `setAttribute()` method to assign the value to the attribute whose name matches the name of the SELECT element.

Listing 19-2: **Controlling HR Object Properties**

```
<HTML>
<HEAD>
<TITLE>HR Object Properties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// document.all normalization trick for NN6
if (navigator.appName == "Netscape" && parseInt(navigator.appVersion) >= 5) {
   document.all = document.getElementsByTagName("*")
}

// one function does all!
function setHRAttr(select) {
```

```
    if (document.all && document.all.myHR) {
        var choice = select.options[select.selectedIndex].value
        if (choice) {
            document.all.myHR.setAttribute(select.name, choice)
        }
    }
}
</SCRIPT>
</HEAD>

<BODY>
<H1>HR Object Properties</H1>
<BR>
<P>Here is the HR element you will be controlling:</P>
<HR ID="myHR">
<FORM>
Select an alignment:
<SELECT NAME="align" onChange="setHRAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="left">Left</OPTION>
    <OPTION VALUE="center">Center</OPTION>
    <OPTION VALUE="right">Right</OPTION>
</SELECT>
<BR>
Select a rule color (IE only):
<SELECT NAME="color" onChange="setHRAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="red">Red</OPTION>
    <OPTION VALUE="green">Green</OPTION>
    <OPTION VALUE="blue">Blue</OPTION>
    <OPTION VALUE="#FA8072">Some Hex Triplet Value</OPTION>
</SELECT>
<BR>
Select a rule shading:
<SELECT NAME="noShade" onChange="setHRAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE=true>No Shading</OPTION>
    <OPTION VALUE=false>Shading</OPTION>
</SELECT>
<BR>
Select a rule height:
<SELECT NAME="size" onChange="setHRAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE=2>2 (Default)</OPTION>
    <OPTION VALUE=4>4 Pixels</OPTION>
    <OPTION VALUE=10>10 Pixels</OPTION>
</SELECT>
<BR>
Select a rule width:
<SELECT NAME="width" onChange="setHRAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="100%">100% (Default)</OPTION>
```

Continued

Listing 19-2 *(continued)*

```
    <OPTION VALUE="80%">80%</OPTION>
    <OPTION VALUE=300>300 Pixels </OPTION>
</SELECT>
</BODY>
</HTML>
```

color

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See Listing 19-2 earlier in this chapter for an example of values that can be used to set the `color` property of an HR element object.

noShade

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See Listing 19-2 earlier in this chapter for an example of values that can be used to set the `noShade` property of an HR element object. Because of the buggy behavior associated with setting this property, adjusting the property in the example has unexpected (and usually undesirable) consequences.

size

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See Listing 19-2 earlier in this chapter for an example of values that can be used to set the `size` property of an HR element object.

width

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See Listing 19-2 earlier in this chapter for an example of values that can be used to set the `width` property of an HR element object.

MARQUEE Element Object

Properties

behavior

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 19-3 contains a page that demonstrates the changes to several MARQUEE element object properties: `behavior`, `bgColor`, `direction`, `scrollAmount`, and `scrollDelay`. This page and scripts are intended only for IE4+. See the description of Listing 19-1 for details on the attribute setting script.

Listing 19-3: Controlling MARQUEE Object Properties

```
<HTML>
<HEAD>
<TITLE>MARQUEE Object Properties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// one function does all!
function setMARQUEEAttr(select) {
    if (document.all && document.all.myMARQUEE) {
        var choice = select.options[select.selectedIndex].value
        if (choice) {
            document.all.myMARQUEE.setAttribute(select.name, choice)
        }
    }
}
```

Continued

Listing 19-3 *(continued)*

```
</SCRIPT>
</HEAD>

<BODY>
<H1>MARQUEE Object Properties</H1>
<BR>
<HR>
<MARQUEE ID="myMARQUEE" WIDTH=400 HEIGHT=24>This is the MARQUEE element object
you will be controlling.</MARQUEE>
<FORM>
<INPUT TYPE="button" VALUE="Start Marquee"
onClick="document.all.myMARQUEE.start()">
<INPUT TYPE="button" VALUE="Stop Marquee"
onClick="document.all.myMARQUEE.stop()">
<BR>
Select a behavior:
<SELECT NAME="behavior" onChange="setMARQUEEAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="alternate">Alternate</OPTION>
    <OPTION VALUE="scroll">Scroll</OPTION>
    <OPTION VALUE="slide">Slide</OPTION>
</SELECT>
<BR>
Select a background color:
<SELECT NAME="bgColor" onChange="setMARQUEEAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="red">Red</OPTION>
    <OPTION VALUE="green">Green</OPTION>
    <OPTION VALUE="blue">Blue</OPTION>
    <OPTION VALUE="#FA8072">Some Hex Triplet Value</OPTION>
</SELECT>
<BR>
Select a scrolling direction:
<SELECT NAME="direction" onChange="setMARQUEEAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE="left">Left</OPTION>
    <OPTION VALUE="right">Right</OPTION>
    <OPTION VALUE="up">Up</OPTION>
    <OPTION VALUE="down">Down</OPTION>
</SELECT>
<BR>
Select a scroll amount:
<SELECT NAME="scrollAmount" onChange="setMARQUEEAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE=4>4</OPTION>
    <OPTION VALUE=6>6 (Default)</OPTION>
    <OPTION VALUE=10>10</OPTION>
</SELECT>
<BR>
Select a scroll delay:
```

```
<SELECT NAME="scrollDelay" onChange="setMARQUEEAttr(this)">
    <OPTION></OPTION>
    <OPTION VALUE=50>Short</OPTION>
    <OPTION VALUE=85>Normal</OPTION>
    <OPTION VALUE=125>Long</OPTION>
</SELECT>
</BODY>
</HTML>
```

bgColor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-3 earlier in this chapter for an example of how to apply values to the `bgColor` property.

direction

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-3 earlier in this chapter for an example of how to apply values to the `direction` property.

scrollAmount
scrollDelay

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-3 earlier in this chapter for an example of how to apply values to the `scrollAmount` and `scrollDelay` properties.

Methods

```
start()
stop()
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-3 earlier in this chapter for examples of both the `start()` and `stop()` methods, which are invoked in event handlers of separate controlling buttons on the page. Notice, too, that when you have the behavior set to `slide`, stopping and restarting the MARQUEE does not cause the scroll action to start from a blank region.

Range Object

Properties

```
collapsed
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `collapsed` property. Reload the page and assign a new range to the `a` global variable by typing the following statement into the top text box:

```
a = document.createRange()
```

Next, set the range to encompass a node:

```
a.selectNode(document.body)
```

Enter `a.collapsed` into the top text box . The expression returns `false` because the end points of the range are not the same.

commonAncestorContainer

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `commonAncestorContainer` property. Reload the page and assign a new range to the `a` global variable by typing the following statement into the top text box:

```
a = document.createRange()
```

Now set the start point to the beginning of the contents of the `myEM` element and set the end point to the end of the surrounding `myP` element:

```
a.setStartBefore(document.getElementById("myEM").firstChild)
a.setEndAfter(document.getElementById("myP").lastChild)
```

Verify that the text range is set to encompass content from the `myEM` node (the word "all") and end of `myP` nodes:

```
a.toString()
```

Verify, too, that the two end point containers are different nodes:

```
a.startContainer.tagName
a.endContainer.tagName
```

Finally, see what node contains both of these two end points:

```
a.commonAncestorContainer.id
```

The result is the `myP` element, which both the `myP` and `myEM` nodes have in common.

endContainer
startContainer

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `endContainer` and `startContainer` properties. Reload the page and assign a new range to the `a` global variable by typing the following statement into the top text box:

```
a = document.createRange()
```

Now set the range to encompass the `myEM` element:

```
a.selectNode(document.getElementById("myEM")
```

Inspect the containers for both the start and end points of the selection:

```
a.startContainer.id
a.endContainer.id
```

The range encompasses the entire `myEM` element, so the start and end points are outside of the element. Therefore, the container of both start and end points is the `myP` element that also surrounds the `myEM` element.

endOffset
startOffset

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `endOffset` and `startOffset` properties, following similar paths you just saw in the description. Reload the page and assign a new range to the `a` global variable by typing the following statement into the top text box:

```
a = document.createRange()
```

Now set the range to encompass the `myEM` element and then move the start point outward to a character within the `myP` element's text node:

```
a.selectNode(document.getElementById("myEM"))
a.setStart(document.getElementById("myP").firstChild, 7)
```

Inspect the node types of the containers for both the start and end points of the selection:

```
a.startContainer.nodeType
a.endContainer.nodeType
```

The `startContainer` node type is 3 (text node), while the `endContainer` node type is 1 (element). Now inspect the offsets for both the start and end points of the selection:

```
a.startOffset
a.endOffset
```

Methods

cloneContents()
cloneRange()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

When Netscape outfits the NN6 browser with the `cloneContents()` method, use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see the method in action. Begin by creating a new range object that contains the text of the `myP` paragraph element.

```
a = document.createRange()
a.selectNode(document.getElementById("myP")
```

Next, clone the original range and preserve the copy in variable `b`:

```
b = a.cloneContents()
```

Move the original range so that it is an insertion point at the end of the body by first expanding it to encompass the entire body and then collapse it to the end

```
a.selectNode(document.body)
a.collapse(false)
```

Now, insert the copy at the very end of the body:

```
a.insertNode(b)
```

If you scroll to the bottom of the page, you see a copy of the text.

See the description of the `compareBoundaryPoints()` method later in this chapter to see an example of the `cloneRange()` method.

collapse([startBoolean])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

See Listings 19-11 (in this chapter) and 15-14 (in Chapter 1 of this book) to see the `collapse()` method at work (albeit with the IE `TextRange` object).

compareBoundaryPoints(*typeInteger*, *sourceRangeRef*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

The page rendered by Listing 19-4 lets you experiment with text range comparisons in NN6+. The bottom paragraph contains a SPAN element that has a `Range` object assigned to its nested text node after the page loads (in the `init()` function). That fixed range becomes a solid reference point for you to use while you select text in the paragraph.

Unfortunately, the `window` object method that converts a user selection into an object is not connected correctly in the first release of NN6. Even if it were, the inverted values returned by the `compareBoundaryPoints()` method would give you incorrect results. Try this example on subsequent versions of NN6.

After you make a selection, all four versions of the `compareBoundaryPoints()` method run to compare the start and end points of the fixed range against your selection. One column of the results table shows the raw value returned by the `compareBoundaryPoints()` method, while the third column puts the results into plain language.

To see how this page works, begin by selecting the first word of the fixed text range (carefully drag the selection from the first red character). You can see that the starting positions of both ranges are the same, because the returned value is `0`. Because all of the invocations of the `compareBoundaryPoints()` method are on the fixed text range, all comparisons are from the point of view of that range. Thus, the first row of the table for the `START_TO_END` parameter indicates that the start point of the fixed range comes before the end point of the selection, yielding a return value of `-1`.

Other selections to make include:

- ✦ Text that starts before the fixed range and ends inside the range
- ✦ Text that starts inside the fixed range and ends beyond the range
- ✦ Text that starts and ends precisely at the fixed range boundaries
- ✦ Text that starts and ends before the fixed range
- ✦ Text that starts after the fixed range

Study the returned values and the plain language results and see how they align with the selection you made.

Listing 19-4: **Lab for NN6 compareBoundaryPoints() Method**

```
<HTML>
<HEAD>
<TITLE>TextRange.compareBoundaryPoints() Method</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
.propName {font-family:Courier, monospace}
#fixedRangeElem {color:red; font-weight:bold}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
var fixedRange

function setAndShowRangeData() {
    try {
        var selectedRange = window.getSelection()
        selectedRange = selectedRange.getRangeAt(0)
        var result1 = fixedRange.compareBoundaryPoints(Range.START_TO_END,
            selectedRange)
        var result2 = fixedRange.compareBoundaryPoints(Range.START_TO_START,
            selectedRange)
        var result3 = fixedRange.compareBoundaryPoints(Range.END_TO_START,
            selectedRange)
        var result4 = fixedRange.compareBoundaryPoints(Range.END_TO_END,
            selectedRange)

        document.getElementById("B1").innerHTML = result1
        document.getElementById("compare1").innerHTML = getDescription(result1)
        document.getElementById("B2").innerHTML = result2
        document.getElementById("compare2").innerHTML = getDescription(result2)
        document.getElementById("B3").innerHTML = result3
        document.getElementById("compare3").innerHTML = getDescription(result3)
        document.getElementById("B4").innerHTML = result4
        document.getElementById("compare4").innerHTML = getDescription(result4)
    }
    catch(err) {
        alert("Vital Range object services are not yet implemented in this
browser.")
    }
}

function getDescription(comparisonValue) {
    switch (comparisonValue) {
        case -1 :
            return "comes before"
            break
        case 0 :
            return "is the same as"
            break
        case 1 :
```

Continued

Listing 19-4 *(continued)*

```
            return "comes after"
            break
        default :
            return "vs."
    }
}

function init() {
    fixedRange = document.createRange()
    fixedRange.selectNodeContents(document.getElementById("fixedRangeElem").
firstChild)
    fixedRange.setEnd(fixedRange.endContainer,
fixedRange.endContainer.nodeValue.length)
}
</SCRIPT>
</HEAD>

<BODY onLoad="init()">
<H1>TextRange.compareBoundaryPoints() Method</H1>
<HR>
<P>Select text in the paragraph in various places relative to
the fixed text range (shown in red). See the relations between
the fixed and selected ranges with respect to their start
and end points.</P>
<TABLE ID="results" BORDER=1 CELLSPACING=2 CELLPADDING=2>
<TR><TH>Property</TH><TH>Returned Value</TH><TH>Fixed Range vs. Selection</TR>
<TR>
    <TD CLASS="propName">StartToEnd</TD>
    <TD CLASS="count" ID="B1"> </TD>
    <TD CLASS="count" ID="C1">Start of Fixed <SPAN ID="compare1">vs.</SPAN>
    End of Selection</TD>
</TR>
<TR>
    <TD CLASS="propName">StartToStart</TD>
    <TD CLASS="count" ID="B2"> </TD>
    <TD CLASS="count" ID="C2">Start of Fixed <SPAN ID="compare2">vs.</SPAN>
    Start of Selection</TD>
</TR>
<TR>
    <TD CLASS="propName">EndToStart</TD>
    <TD CLASS="count" ID="B3"> </TD>
    <TD CLASS="count" ID="C3">End of Fixed <SPAN ID="compare3">vs.</SPAN>
    Start of Selection</TD>
</TR>
<TR>
    <TD CLASS="propName">EndToEnd</TD>
    <TD CLASS="count" ID="B4"> </TD>
    <TD CLASS="count" ID="C4">End of Fixed <SPAN ID="compare4">vs.</SPAN>
    End of Selection</TD>
</TR>
```

```
</TABLE>
<HR>
<P onMouseUp="setAndShowRangeData()">
Lorem ipsum dolor sit, <SPAN ID="fixedRangeElem">consectetaur adipisicing
elit</SPAN>,
sed do eiusmod tempor incididunt ut labore et dolore aliqua. Ut enim adminim
veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat.</P>
</BODY>
</HTML>
```

createContextualFragment("*text*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to create a document fragment and replace an existing document tree node with the fragment. Begin by creating the range and fragment:

```
a = document.createRange()
a.selectNode(document.body)
b = a.createContextualFragment("<SPAN STYLE='font-size:22pt'>a bunch of
</SPAN>")
```

This fragment consists of a SPAN element node with a text node nested inside. At this point, you can inspect the properties of the document fragment by entering `b` into the bottom text box.

To replace the `myEM` element on the page with this new fragment, use the `replaceChild()` method on the enclosing `myP` element:

```
document.getElementById("myP").replaceChild(b, document.getElementById("myEM"))
```

The fragment now becomes a legitimate child node of the `myP` element and can be referenced like any node in the document tree. For example, if you enter the following statement into the top text box of The Evaluator, you can retrieve a copy of the text node inside the new SPAN element:

```
document.getElementById("myP").childNodes[1].firstChild.nodeValue
```

deleteContents()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with deleting contents of both a text node and a complete element node. Begin by creating a text range for the text node inside the `myEM` element (enter the third statement, which wraps below, as one continuous expression):

```
a = document.createRange()
a.setStart(document.getElementById("myEM").firstChild, 0)
a.setEnd(document.getElementById("myEM").lastChild,
    document.getElementById("myEM").lastChild.length)
```

Verify the makeup of the range by entering `a` into the bottom text box and inspect its properties. Both containers are text nodes (they happen to be the same text node), and offsets are measured by character positions.

Now, delete the contents of the range:

```
a.deleteContents()
```

The italicized word "all" is gone from the tree, but the `myEM` element is still there. To prove it, put some new text inside the element:

```
document.getElementById("myEM").innerHTML = "a band of "
```

The italic style of the EM element applies to the text, as it should.

Next, adjust the range boundaries to include the `myEM` element tags, as well:

```
a.selectNode(document.getElementById("myEM"))
```

Inspect the `Range` object's properties again by entering `a` into the bottom text box. The container nodes are the P element that surrounds the EM element; the offset values are measured in nodes. Delete the range's contents:

```
a.deleteContents()
```

Not only is the italicized text gone, but the `myEM` element is gone, too. The `myP` element now has but one child node, the text node inside. The following entries into the top text box of The Evaluator verify this fact:

```
document.getElementById("myP").childNodes.length
document.getElementById("myP").childNodes[0].nodeValue
```

If you try this example in early versions of NN6, however, you see that the `deleteContents()` method also removes the text node following the `myEM` element. This is buggy behavior, demonstrating that the method works best on text nodes, rather than elements.

extractContents()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

When Netscape outfits the NN6 browser with the `extractContents()` method, use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see how the method works. Begin by creating a new range object that contains the text of the `myP` paragraph element.

```
a = document.createRange()
a.selectNode(document.getElementById("myP"))
```

Next, extract the original range's content and preserve the copy in variable `b`:

```
b = a.extractContents()
```

Move the original range so that it is an insertion point at the end of the body by first expanding it to encompass the entire body and then collapse it to the end

```
a.selectNode(document.body)
a.collapse(false)
```

Now, insert the extracted fragment at the very end of the body:

```
a.insertNode(b)
```

If you scroll to the bottom of the page, you see a copy of the text.

insertNode(*nodeReference*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Listing 19-5, which relies on `selection` and `Range` object features not implemented in the first release of NN6, demonstrates the `insertNode()` method plus some additional items from the NN6 `selection` object. The example even includes a rudimentary undo buffer for scripted changes to a text range. In the page generated by this listing, users can select any text in a paragraph and have the script automatically convert the text to all uppercase characters. The task of replacing a selection with other text requires several steps, starting with the selection, which is retrieved via the `window.getSelection()` method. After making sure the selection contains some text (that is, the selection isn't collapsed), the selection is preserved as a range object so that the starting text can be stored in a global variable (as a

property of the `undoBuffer` global variable object). After that, the selection is deleted from the document tree, leaving the selection as a collapsed insertion point. A copy of that selection in the form of a range object is preserved in the `undoBuffer` object so that the undo script knows where to reinsert the original text. A new text node is created with an uppercase version of the original text, and, finally, the `insertNode()` method is invoked to stick the converted text into the collapsed range.

Undoing this operation works in reverse. Original locations and strings are copied from the `undoBuffer` object. After creating the range with the old start and end points (which represent a collapsed insertion point), the resurrected text (converted to a text node) is inserted into the collapsed range. For good housekeeping, the `undoBuffer` object is restored to its unused form.

Listing 19-5: **Inserting a Node into a Range**

```
<HTML>
<HEAD>
<TITLE>NN Selection Object Replacement</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var undoBuffer = {rng:null, txt:""}
function convertSelection() {
    var sel, grossRng, netRng, newText
    try {
        sel = window.getSelection()
        if (!sel.isCollapsed) {
            grossRng = sel.getRangeAt(0)
            undoBuffer.txt = grossRng.toString()
            sel.deleteFromDocument()
            netRng = sel.getRangeAt(0)
            undoBuffer.rng = netRng
            newText = document.createTextNode(undoBuffer.txt.toUpperCase())
            netRng.insertNode(newText)
        }
    }
    catch(err) {
        alert("Vital Range object services are not yet implemented in this
browser.")
    }
}
function undoConversion() {
    var rng, oldText
    if (undoBuffer.rng) {
        rng = document.createRange()
        rng.setStart(undoBuffer.rng.startParent, undoBuffer.rng.startOffset)
        rng.setEnd(undoBuffer.rng.endParent, undoBuffer.rng.endOffset)
        oldText = document.createTextNode(undoBuffer.txt)
        rng.insertNode(oldText)
        undoBuffer.rng = null
        undoBuffer.txt = ""
    }
}
```

```
</SCRIPT>
</HEAD>
<BODY>
<H1 ID="H1_1">NN6 Selection Object Replacement</H1>
<HR>
<P ID="P_1" onMouseUp="convertSelection()">This paragraph
contains text that you can select. Selections are deleted and
replaced by all uppercase versions of the selected text.</P>
<BUTTON onClick="undoConversion()">Undo Last</BUTTON>
<BUTTON onClick="location.reload(true)">Start Over</BUTTON>
</BODY>
</HTML>
```

isValidFragment("*HTMLText*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

You can try the validity of any strings that you like in The Evaluator (Chapter 13 in the *JavaScript Bible*). You will discover, however, that the object model can make a document fragment out of just about any string. For instance, if you attempt to create a document fragment out of some random text and an end tag, the document fragment will consist of a text node and an element node of the type indicated by the end tag.

selectNode(*nodeReference*)
selectNodeContents(*nodeReference*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see the behavior of both the `selectNode()` and `selectNodeContents()` methods work. Begin by creating a new range object.

```
a = document.createRange()
```

Set the range boundaries to include the `myP` element node:

```
a.selectNode(document.getElementById("myP"))
```

Enter `a` into the bottom text box to view the properties of the range. Notice that because the range has selected the entire paragraph node, the container of the range's start and end points is the BODY element of the page (the parent element of the `myP` element).

Now change the range so that it encompasses only the contents of the `myP` element:

```
a.selectNodeContents(document.getElementById("myP"))
```

Click the List Properties button to view the current properties of the range. The container of the range's boundary points is the P element that holds the element's contents.

setEnd(*nodeReference, offset*)
setStart(*nodeReference, offset*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with both the `setStart()` and `setEnd()` methods. Begin by creating a new range object.

```
a = document.createRange()
```

For the first range, set the start and end points to encompass the second node (the `myEM` element) inside the `myP` element:

```
a.setStart(document.getElementById("myP"), 1)
a.setEnd(document.getElementById("myP"), 2)
```

The text encompassed by the range consists of the word "all" plus the trailing space that is contained by the `myEM` element. Prove this by entering the following statement into the top text box:

```
a.toString()
```

If you then click the Results box to the right of the word "all," you see that the results contain the trailing space. Yet, if you examine the properties of the range (enter `a` into the bottom text box), you see that the range is defined as actually starting before the `myEM` element and ending after it.

Next, adjust the start point of the range to a character position inside the first text node of the `myP` element:

```
a.setStart(document.getElementById("myP").firstChild, 11)
```

Click the List Properties button to see that the `startContainer` property of the range is the text node, and that the `startOffset` measures the character position. All end boundary properties, however, have not changed. Enter `a.toString()` in the top box again to see that the range now encompasses text from two of the nodes inside the `myP` element.

You can continue to experiment by setting the start and end points to other element and text nodes on the page. After each adjustment, verify the properties of the `a` range object and the text it encompasses (via `a.toString()`).

setEndAfter(*nodeReference*)
setEndBefore(*nodeReference*)
setStartAfter(*nodeReference*)
setStartBefore(*nodeReference*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with all four methods. Begin by creating a new range object.

```
a = document.createRange()
```

For the first range, set the start and end points to encompass the `myEM` element inside the `myP` element:

```
a.setStartBefore(document.getElementById("myEM"))
a.setEndAfter(document.getElementById("myEM"))
```

The text encompassed by the range consists of the word "all" plus the trailing space that is contained by the `myEM` element. Prove this by entering the following statement into the top text box:

```
a.toString()
```

Next, adjust the start point of the range to the beginning of the first text node of the `myP` element:

```
a.setStartBefore(document.getElementById("myP").firstChild)
```

Enter `a` into the bottom text box to see that the `startParent` property of the range is the `P` element node, while the `endParent` property points to the EM element.

You can continue to experiment by setting the start and end points to before and after other element and text nodes on the page. After each adjustment, verify the properties of the `a` range object and the text it encompasses (via `a.toString()`).

surroundContents(*nodeReference*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Listing 19-6, which relies on `selection` and `Range` object features not implemented in the first release of NN6, demonstrates how the `surroundContents()` method wraps a range inside a new element. As the page loads, a global variable (`newSpan`) stores a SPAN element that is used as a prototype for elements to be used as new surrounding parent nodes. When you select text in either of the two paragraphs, the selection is converted to a range. The `surroundContents()` method then wraps the range with the `newSpan` element. Because that SPAN element has a class name of `hilite`, the element and its contents pick up the style sheet properties as defined for that class selector.

Listing 19-6: **Using the Range.surroundContents() Method**

```
<HTML>
<HEAD>
<TITLE>Range.surroundContents() Method</TITLE>
<STYLE TYPE="text/css">
.hilite {background-color:yellow; color:red; font-weight:bold}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
var newSpan = document.createElement("SPAN")
newSpan.className = "hilite"

function highlightSelection() {
    var sel, rng
    try {
        sel = window.getSelection()
        if (!sel.isCollapsed) {
            rng = sel.getRangeAt(0)
            rng.surroundContents(newSpan.cloneNode(false))
        }
    }
    catch(err) {
        alert("Vital Range object services are not yet implemented in this
browser.")
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Range.surroundContents() Method</H1>
<HR>
<P onMouseUp="highlightSelection()">These paragraphs
contain text that you can select. Selections are surrounded
by SPAN elements that share a stylesheet class selector
for special font and display characteristics.</P>

<P onMouseUp="highlightSelection()">Lorem ipsum dolor
sit amet, consectetaur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
```

```
aliqua. Ut enim adminim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.</P>
</BODY>
</HTML>
```

toString()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see the results of the `toString()` method. Enter the following sequence of statements into the top text box:

```
a = document.createRange()
a.selectNode(document.getElementById("myP"))
a.toString()
```

If you type only `a` into the top text box, you see the text contents of the range, but don't be fooled. Internal workings of The Evaluator attempt to evaluate any expression entered into that text field. Assigning a range object to a text box forces an internal application of the `toString()` method (just as the `Date` object does when you create a new object instance in The Evaluator).

selection Object

Properties

type

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 19-7 contains a page that demonstrates several features of the `selection` object. When you make a selection with the Deselect radio button selected, you see the value of the `selection.type` property (in the statusbar) before and after the selection is deselected. After the selection goes away, the `type` property returns `None`.

Listing 19-7: **Using the document.selection Object**

```
<HTML>
<HEAD>
<TITLE>selection Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function processSelection() {
    if (document.choices.process[0].checked) {
        status = "Selection is type: " + document.selection.type
        setTimeout("emptySelection()", 2000)
    } else if (document.choices.process[1].checked) {
        var rng = document.selection.createRange()
        document.selection.clear()
    }
}
function emptySelection() {
    document.selection.empty()
    status = "Selection is type: " + document.selection.type
}
</SCRIPT>
</HEAD>
<BODY>
<H1>IE selection Object</H1>
<HR>
<FORM NAME="choices">
<INPUT TYPE="radio" NAME="process" CHECKED>De-select after two seconds<BR>
<INPUT TYPE="radio" NAME="process">Delete selected text.
</FORM>
<P onMouseUp="processSelection()">Lorem ipsum dolor sit amet, consectetaur
adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim adminim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit
involuptate velit esse cillum dolore eu fugiat nulla pariatur.
</BODY>
</HTML>
```

Methods

clear()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-7 earlier in this chapter to see the `selection.clear()` method at work.

`createRange()`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listings 15-36 and 15-45 to see the `selection.createRange()` method turn user selections into text ranges.

`empty()`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-7 earlier in this chapter to view the `selection.empty()` method at work.

Text and TextNode Objects

Properties

`data`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

In the example for the `nodeValue` property used in a text replacement script (in Chapter 1 of this book), you can substitute the `data` property for `nodeValue` to accomplish the same result.

Methods

```
appendData("text")
deleteData(offset, count)
insertData(offset, "text")
replaceData(offset, count, "text")
substringData(offset, count)
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

The page created by Listing 19-8 is a working laboratory that you can use to experiment with the five data-related methods in NN6+. The text node that invokes the methods is a simple sentence in a P element. Each method has its own clickable button, followed by two or three text boxes into which you enter values for method parameters. Don't be put off by the length of the listing. Each method's operation is confined to its own function and is fairly simple.

Each of the data-related methods throws exceptions of different kinds. To help handle these errors gracefully, the method calls are wrapped inside a `try/catch` construction. All caught exceptions are routed to the `handleError()` function where details of the error are inspected and friendly alert messages are displayed to the user. See Chapter 39 of the *JavaScript Bible* for details on the `try/catch` approach to error handling in W3C DOM-capable browsers.

Listing 19-8: **Text object Data Method Laboratory**

```
<HTML>
<HEAD>
<TITLE>Data Methods of a W3C Text Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doAppend(form) {
    var node = document.getElementById("myP").firstChild
    var newString = form.appendStr.value
    try {
        node.appendData(newString)
    }
    catch(err) {
        handleError(err)
    }
}
function doDelete(form) {
    var node = document.getElementById("myP").firstChild
    var offset = form.deleteOffset.value
```

```
    var count = form.deleteCount.value
    try {
        node.deleteData(offset, count)
    }
    catch(err) {
        handleError(err)
    }
}
function doInsert(form) {
    var node = document.getElementById("myP").firstChild
    var offset = form.insertOffset.value
    var newString = form.insertStr.value
    try {
        node.insertData(offset, newString)
    }
    catch(err) {
        handleError(err)
    }

}

function doReplace(form) {
    var node = document.getElementById("myP").firstChild
    var offset = form.replaceOffset.value
    var count = form.replaceCount.value
    var newString = form.replaceStr.value
    try {
        node.replaceData(offset, count, newString)
    }
    catch(err) {
        handleError(err)
    }
}
function showSubstring(form) {
    var node = document.getElementById("myP").firstChild
    var offset = form.substrOffset.value
    var count = form.substrCount.value
    try {
        alert(node.substringData(offset, count))
    }
    catch(err) {
        handleError(err)
    }
}
// error handler for these methods
function handleError(err) {
    switch (err.name) {
        case "NS_ERROR_DOM_INDEX_SIZE_ERR":
            alert("The offset number is outside the allowable range.")
            break
        case "NS_ERROR_DOM_NOT_NUMBER_ERR":
            alert("Make sure each numeric entry is a valid number.")
```

Continued

Listing 19-8 *(continued)*

```
            break
        default:
            alert("Double-check your text box entries.")
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Data Methods of a W3C Text Object</H1>
<HR>
<P ID="myP" STYLE="font-weight:bold; text-align:center">
So I called myself Pip, and became to be called Pip.</P>
<FORM NAME="choices">
<P><INPUT TYPE="button" onClick="doAppend(this.form)" VALUE="appendData()">
String:<INPUT TYPE="text" NAME="appendStr" SIZE=30></P>

<P><INPUT TYPE="button" onClick="doDelete(this.form)" VALUE="deleteData()">
Offset:<INPUT TYPE="text" NAME="deleteOffset" SIZE=3>
Count:<INPUT TYPE="text" NAME="deleteCount" SIZE=3></P>

<P><INPUT TYPE="button" onClick="doInsert(this.form)" VALUE="insertData()">
Offset:<INPUT TYPE="text" NAME="insertOffset" SIZE=3>
String:<INPUT TYPE="text" NAME="insertStr" SIZE=30></P>

<P><INPUT TYPE="button" onClick="doReplace(this.form)" VALUE="replaceData()">
Offset:<INPUT TYPE="text" NAME="replaceOffset" SIZE=3>
Count:<INPUT TYPE="text" NAME="replaceCount" SIZE=3>
String:<INPUT TYPE="text" NAME="replaceStr" SIZE=30></P>

<P><INPUT TYPE="button" onClick="showSubstring(this.form)"
VALUE="substringData()">
Offset:<INPUT TYPE="text" NAME="substrOffset" SIZE=3>
Count:<INPUT TYPE="text" NAME="substrCount" SIZE=3></P>

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

splitText(*offset*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see the `splitText()` method in action. Begin by verifying that the `myEM` element has but one child node, and that its `nodeValue` is the string "all":

```
document.getElementById("myEM").childNodes.length
document.getElementById("myEM").firstChild.nodeValue
```

Next, split the text node into two pieces after the first character:

```
document.getElementById("myEM").firstChild.splitText(1)
```

Two text nodes are now inside the element:

```
document.getElementById("myEM").childNodes.length
```

Each text node contains its respective portion of the original text:

```
document.getElementById("myEM").firstChild.nodeValue
document.getElementById("myEM").lastChild.nodeValue
```

If you are using NN6, now bring the text nodes back together:

```
document.getElementById("myEM").normalize()
document.getElementById("myEM").childNodes.length
```

At no time during these statement executions does the rendered text change.

TextRange Object

Properties

`boundingHeight`
`boundingLeft`
`boundingTop`
`boundingWidth`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 19-9 provides a simple playground to explore the four bounding properties (and two offset properties) of a `TextRange` object. As you select text in the big paragraph, the values of all six properties are displayed in the table. Values are also updated if you resize the window via an `onResize` event handler.

Notice, for example, if you simply click in the paragraph without dragging a selection, the `boundingWidth` property shows up as zero. This action is the equivalent of a `TextRange` acting as an insertion point.

Listing 19-9: **Exploring the Bounding TextRange Properties**

```
<HTML>
<HEAD>
<TITLE>TextRange Object Dimension Properties</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
.propName {font-family: Courier, monospace}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function setAndShowRangeData() {
    var range = document.selection.createRange()
    B1.innerText = range.boundingHeight
    B2.innerText = range.boundingWidth
    B3.innerText = range.boundingTop
    B4.innerText = range.boundingLeft
    B5.innerText = range.offsetTop
    B6.innerText = range.offsetLeft
}
</SCRIPT>
</HEAD>

<BODY onResize="setAndShowRangeData()">
<H1>TextRange Object Dimension Properties</H1>
<HR>
<P>Select text in the paragraph below and observe the "bounding"
property values for the TextRange object created for that selection.</P>
<TABLE ID="results" BORDER=1 CELLSPACING=2 CELLPADDING=2>
<TR><TH>Property</TH><TH>Pixel Value</TH></TR>
<TR>
    <TD CLASS="propName">boundingHeight</TD>
    <TD CLASS="count" ID="B1"> </TD>
</TR>
<TR>
    <TD CLASS="propName">boundingWidth</TD>
    <TD CLASS="count" ID="B2"> </TD>
</TR>
<TR>
    <TD CLASS="propName">boundingTop</TD>
    <TD CLASS="count" ID="B3"> </TD>
</TR>
<TR>
    <TD CLASS="propName">boundingLeft</TD>
    <TD CLASS="count" ID="B4"> </TD>
</TR>
<TR>
    <TD CLASS="propName">offsetTop</TD>
    <TD CLASS="count" ID="B5"> </TD>
</TR>
<TR>
    <TD CLASS="propName">offsetLeft</TD>
    <TD CLASS="count" ID="B6"> </TD>
```

```
</TR>
</TABLE>
<HR>
<P onMouseUp="setAndShowRangeData()">
Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim adminim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit involuptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deseruntmollit anim id est laborum
Et harumd und lookum like Greek to me, dereud facilis est er expedit.
</P>
</BODY>
</HTML>
```

htmlText

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to investigate values returned by the `htmlText` property. Use the top text box to enter the following statements and see the values in the Results box.

Begin by creating a `TextRange` object for the entire body and store the range in local variable `a`:

```
a = document.body.createTextRange()
```

Next, use the `findText()` method to set the start and end points of the text range around the word "all," which is an EM element inside the `myP` paragraph:

```
a.findText("all")
```

The method returns `true` (see the `findText()` method) if the text is found and the text range adjusts to surround it. To prove that the text of the text range is what you think it is, examine the `text` property of the range:

```
a.text
```

Because the text range encompasses all of the text of the element, the `htmlText` property contains the tags for the element as well:

```
a.htmlText
```

If you want to experiment by finding other chunks of text and looking at both the `text` and `htmlText` properties, first restore the text range to encompass the entire body with the following statement:

```
a.expand("textEdit")
```

You can read about the `expand()` method later in this chapter. In other tests, use `findText()` to set the range to "for all" and just "for al." Then, see how the `htmlText` property exposes the EM element's tags.

text

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-11 later in this chapter for the `findText()` method to see the `text` property used to perform the replace action of a search-and-replace function.

Methods

collapse([*startBoolean*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listings 19-11 (in this chapter) and 15-14 (in Chapter 1 of this book) to see the `collapse()` method at work.

compareEndPoints("*type*", *rangeRef*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

The page rendered by Listing 19-10 lets you experiment with text range comparisons. The bottom paragraph contains a SPAN element that has a `TextRange` object assigned to its text after the page loads (in the `init()` function). That fixed range becomes a solid reference point for you to use while you select text in the

paragraph. After you make a selection, all four versions of the `compareEndPoints()` method run to compare the start and end points of the fixed range against your selection. One column of the results table shows the raw value returned by the `compareEndPoints()` method, while the third column puts the results into plain language.

To see how this page works, begin by selecting the first word of the fixed text range (double-click the word). You can see that the starting positions of both ranges are the same, because the returned value is `0`. Because all of the invocations of the `compareEndPoints()` method are on the fixed text range, all comparisons are from the point of view of that range. Thus, the first row of the table for the `StartToEnd` parameter indicates that the start point of the fixed range comes before the end point of the selection, yielding a return value of `-1`.

Other selections to make include:

- ✦ Text that starts before the fixed range and ends inside the range
- ✦ Text that starts inside the fixed range and ends beyond the range
- ✦ Text that starts and ends precisely at the fixed range boundaries
- ✦ Text that starts and ends before the fixed range
- ✦ Text that starts after the fixed range

Study the returned values and the plain language results and see how they align with the selection you make.

Listing 19-10: **Lab for compareEndPoints() Method**

```
<HTML>
<HEAD>
<TITLE>TextRange.compareEndPoints() Method</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
.propName {font-family:Courier, monospace}
#fixedRangeElem {color:red; font-weight:bold}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
var fixedRange

function setAndShowRangeData() {
    var selectedRange = document.selection.createRange()
    var result1 = fixedRange.compareEndPoints("StartToEnd", selectedRange)
    var result2 = fixedRange.compareEndPoints("StartToStart", selectedRange)
    var result3 = fixedRange.compareEndPoints("EndToStart", selectedRange)
    var result4 = fixedRange.compareEndPoints("EndToEnd", selectedRange)

    B1.innerText = result1
    compare1.innerText = getDescription(result1)
    B2.innerText = result2
    compare2.innerText = getDescription(result2)
```

Continued

Listing 19-10 *(continued)*

```
    B3.innerText = result3
    compare3.innerText = getDescription(result3)
    B4.innerText = result4
    compare4.innerText = getDescription(result4)
}

function getDescription(comparisonValue) {
    switch (comparisonValue) {
        case -1 :
            return "comes before"
            break
        case 0 :
            return "is the same as"
            break
        case 1 :
            return "comes after"
            break
        default :
            return "vs."
    }
}

function init() {
    fixedRange = document.body.createTextRange()
    fixedRange.moveToElementText(fixedRangeElem)
}
</SCRIPT>
</HEAD>

<BODY onLoad="init()">
<H1>TextRange.compareEndPoints() Method</H1>
<HR>
<P>Select text in the paragraph in various places relative to
the fixed text range (shown in red). See the relations between
the fixed and selected ranges with respect to their start
and end points.</P>
<TABLE ID="results" BORDER=1 CELLSPACING=2 CELLPADDING=2>
<TR><TH>Property</TH><TH>Returned Value</TH><TH>Fixed Range vs. Selection</TR>
<TR>
    <TD CLASS="propName">StartToEnd</TD>
    <TD CLASS="count" ID="B1"> </TD>
    <TD CLASS="count" ID="C1">Start of Fixed
    <SPAN ID="compare1">vs.</SPAN> End of Selection</TD>
</TR>
<TR>
    <TD CLASS="propName">StartToStart</TD>
    <TD CLASS="count" ID="B2"> </TD>
    <TD CLASS="count" ID="C2">Start of Fixed
    <SPAN ID="compare2">vs.</SPAN> Start of Selection</TD>
</TR>
```

```
<TR>
    <TD CLASS="propName">EndToStart</TD>
    <TD CLASS="count" ID="B3"> </TD>
    <TD CLASS="count" ID="C3">End of Fixed
    <SPAN ID="compare3">vs.</SPAN> Start of Selection</TD>
</TR>
<TR>
    <TD CLASS="propName">EndToEnd</TD>
    <TD CLASS="count" ID="B4"> </TD>
    <TD CLASS="count" ID="C4">End of Fixed
    <SPAN ID="compare4">vs.</SPAN> End of Selection</TD>
</TR>
</TABLE>
<HR>
<P onMouseUp="setAndShowRangeData()">
Lorem ipsum dolor sit, <SPAN ID="fixedRangeElem">consectetaur adipisicing
elit</SPAN>,
sed do eiusmod tempor incididunt ut labore et dolore aliqua. Ut enim adminim
veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat.</P>
</BODY>
</HTML>
```

duplicate()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see how the `duplicate()` method works. Begin by creating a new `TextRange` object that contains the text of the `myP` paragraph element.

```
a = document.body.createTextRange()
a.moveToElementText(myP)
```

Next, clone the original range and preserve the copy in variable `b`:

```
b = a.duplicate()
```

The method returns no value, so don't be alarmed by the "undefined" that appears in the Results box. Move the original range so that it is an insertion point at the end of the body by first expanding it to encompass the entire body, and then collapse it to the end:

```
a.expand("textedit")
a.collapse(false)
```

Now, insert the copy at the very end of the body:

```
a.text = b.text
```

If you scroll to the bottom of the page, you'll see a copy of the text.

execCommand("*commandName*"[, *UIFlag*[, *value*]])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see how to copy a text range's text into the client computer's Clipboard. Begin by setting the text range to the `myP` element:

```
a = document.body.createTextRange()
a.moveToElementText(myP)
```

Now use `execCommand()` to copy the range into the Clipboard:

```
a.execCommand("Copy")
```

To prove that the text is in the Clipboard, click the bottom text field and choose Paste from the Edit menu (or press `Ctrl+V`).

expand("*unit*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

You can find examples of the `expand()` method in Listing 15-14.

findText("*searchString*"[, *searchScope*, *flags*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 19-11 implements two varieties of a text search-and-replace operation, while showing you how to include extra parameters for case-sensitive and whole word searches. Both approaches begin by creating a `TextRange` for the entire body, but they immediately shift the starting point to the beginning of the DIV element that contains the text to search.

One search-and-replace function prompts the user to accept or decline replacement for each instance of a found string. The `select()` and `scrollIntoView()` methods are invoked to help the user see what is about to be replaced. Notice that even when the user declines to accept the replacement, the text range is collapsed to the end of the found range so that the next search can begin after the previously found text. Without the `collapse()` method, the search can get caught in an infinite loop as it keeps finding the same text over and over (with no replacement made). Because no counting is required, this search-and-replace operation is implemented inside a `while` repeat loop.

The other search-and-replace function goes ahead and replaces every match and then displays the number of replacements made. After the loop exits (because there are no more matches), the loop counter is used to display the number of replacements made.

Listing 19-11: Two Search and Replace Approaches (with Undo)

```
<HTML>
<HEAD>
<TITLE>TextRange.findText() Method</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// global range var for use with Undo
var rng

// return findText() third parameter arguments
function getArgs(form) {
    var isCaseSensitive = (form.caseSensitive.checked) ? 4 : 0
    var isWholeWord = (form.wholeWord.checked) ? 2 : 0
    return isCaseSensitive ^ isWholeWord
}

// prompted search and replace
function sAndR(form) {
    var srchString = form.searchString.value
    var replString = form.replaceString.value
    if (srchString) {
        var args = getArgs(form)
        rng = document.body.createTextRange()
        rng.moveToElementText(rights)
        clearUndoBuffer()
        while (rng.findText(srchString, 10000, args)) {
            rng.select()
```

Continued

Listing 19-11 *(continued)*

```
            rng.scrollIntoView()
            if (confirm("Replace?")) {
                rng.text = replString
                pushUndoNew(rng, srchString, replString)
            }
            rng.collapse(false)
        }
    }
}

// unprompted search and replace with counter
function sAndRCount(form) {
    var srchString = form.searchString.value
    var replString = form.replaceString.value
    var i
    if (srchString) {
        var args = getArgs(form)
        rng = document.body.createTextRange()
        rng.moveToElementText(rights)
        for (i = 0; rng.findText(srchString, 10000, args); i++) {
            rng.text = replString
            pushUndoNew(rng, srchString, replString)
            rng.collapse(false)
        }
        if (i > 1) {
            clearUndoBuffer()
        }
    }
    document.all.counter.innerText = i
}

// BEGIN UNDO BUFFER CODE
// buffer global variables
var newRanges = new Array()
var origSearchString

// store original search string and bookmarks of each replaced range
function pushUndoNew(rng, srchString, replString) {
    origSearchString = srchString
    rng.moveStart("character", -replString.length)
    newRanges[newRanges.length] = rng.getBookmark()
}

// empty array and search string global
function clearUndoBuffer() {
    document.all.counter.innerText = "0"
    origSearchString = ""
    newRanges.length = 0
}
```

```
// perform the undo
function undoReplace() {
    if (newRanges.length && origSearchString) {
        for (var i = 0; i < newRanges.length; i++) {
            rng.moveToBookmark(newRanges[i])
            rng.text = origSearchString
        }
        document.all.counter.innerText = i
        clearUndoBuffer()
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>TextRange.findText() Method</H1>
<HR>
<FORM>
<P>Enter a string to search for in the following text:
<INPUT TYPE="text" NAME="searchString" SIZE=20 VALUE="Law">  
<INPUT TYPE="checkbox" NAME="caseSensitive">Case-sensitive  
<INPUT TYPE="checkbox" NAME="wholeWord">Whole words only</P>
<P>Enter a string with which to replace found text:
<INPUT TYPE="text" NAME="replaceString" SIZE=20 VALUE="legislation"></P>
<P><INPUT TYPE="button" VALUE="Search and Replace (with prompt)"
onClick="sAndR(this.form)"></P>
<P><INPUT TYPE="button" VALUE="Search, Replace, and Count (no prompt)"
onClick="sAndRCount(this.form)">
<SPAN ID="counter">0</SPAN> items found and replaced.</P>
<P><INPUT TYPE="button" VALUE="Undo Search and Replace"
onClick="undoReplace()"></P>
</FORM>

<DIV ID="rights">
<A NAME="article1">
<H2>ARTICLE I</H2>
</A>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of speech, or of
the press; or the right of the people peaceably to assemble, and to petition the
government for a redress of grievances.
</P>
[The rest of the text is snipped for printing here, but it is on the CD-ROM
version.]
</DIV>
</BODY>
</HTML>
```

Having a search-and-replace function available in a document is only one-half of the battle. The other half is offering the facilities to undo the changes. To that end,

Listing 19-11 includes an undo buffer that accurately undoes only the changes made in the initial replacement actions.

The undo buffer stores its data in two global variables. The first, `origSearchString`, is simply the string used to perform the original search. This variable is the string that has to be put back in the places where it had been replaced. The second global variable is an array that stores `TextRange` bookmarks (see `getBookmark()` later in this chapter). These references are string values that don't mean much to humans, but the browser can use them to recreate a range with its desired start and end points. Values for both the global search string and bookmark specifications are stored in calls to the `pushUndoNew()` method each time text is replaced.

A perhaps unexpected action of setting the `text` property of a text range is that the start and end points collapse to the end of the new text. Because the stored bookmark must include the replaced text as part of its specification, the start point of the current range must be adjusted back to the beginning of the replacement text before the bookmark can be saved. Thus, the `pushUndoNew()` function receives the replacement text string so that the `moveStart()` method can be adjusted by the number of characters matching the length of the replacement string.

After all of the bookmarks are stored in the array, the undo action can do its job in a rather simple `for` loop inside the `undoReplace()` function. After verifying that the undo buffer has data stored in it, the function loops through the array of bookmarks and replaces the bookmarked text with the old string. The benefit of using the bookmarks rather than using the replacement function again is that only those ranges originally affected by the search-and-replace operation are touched in the undo operation. For example, in this document if you replace a case-sensitive "states" with "States" two replacements are performed. At that point, however, the document has four instances of "States," two of which existed before. Redoing the replacement function by inverting the search-and-replace strings would convert all four back to the lowercase version — not the desired effect.

`getBookmark()`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 19-11 earlier in this chapter shows how the `getBookmark()` method is used to preserve specifications for text ranges so that they can be called upon again to be used to undo changes made to the text range. The `getBookmark()` method is used to save the snapshots, while the `moveToBookmark()` method is used during the undo process.

inRange(*otherRangeRef*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see the `inRange()` method in action. The following statements generate two distinct text ranges, one for the `myP` paragraph element and the other for the `myEM` element nested within.

```
a = document.body.createTextRange()
a.moveToElementText(myP)
b = document.body.createTextRange()
b.moveToElementText(myEM)
```

Because the `myP` text range is larger than the other, invoke the `inRange()` method on it, fully expecting the return value of `true`

```
a.inRange(b)
```

But if you switch the references, you see that the larger text range is not "in" the smaller one:

```
b.inRange(a)
```

isEqual(*otherRangeRef*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to try the `isEqual()` method. Begin by creating two separate `TextRange` objects, one for the `myP` element and one for `myEM`.

```
a = document.body.createTextRange()
a.moveToElement(myP)
b = document.body.createTextRange()
b.moveToElement(myEM)
```

Because these two ranges encompass different sets of text, they are not equal, as the results show from the following statement:

```
a.isEqual(b)
```

But if you now adjust the first range boundaries to surround the `myEM` element, both ranges are the same values:

```
a.moveToElement(myEM)
a.isEqual(b)
```

move("*unit*"[, *count*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `move()` method. To see how the method returns just the number of units it moves the pointer, begin by creating a text range and set it to enclose the `myP` element:

```
a = document.body.createTextRange()
a.moveToElementText(myP)
```

Now enter the following statement to collapse and move the range backward by 20 words.

```
a.move("word", -20)
```

Continue to click the Evaluate button and watch the returned value in the Results box. The value shows `20` while it can still move backward by 20 words. But eventually the last movement will be some other value closer to zero. And after the range is at the beginning of the BODY element, the range can move no more in that direction, so the result is zero.

moveEnd("*unit*"[, *count*])
moveStart("*unit*"[, *count*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `moveEnd()` and `moveStart()` methods. Begin by creating a text range and set it to enclose the `myEM` element:

```
a = document.body.createTextRange()
a.moveToElementText(myEM)
```

To help you see how movements of the pointers affect the text enclosed by the range, type `a` into the bottom text box and view all the properties of the text range. Note especially the `htmlText` and `text` properties. Now enter the following statement to move the end of the range forward by one word.

```
a.moveEnd("word")
```

Click on the List Properties button to see that the text of the range now includes the word following the EM element. Try each of the following statements in the top text box and examine both the integer results and (by clicking the List Properties button) the properties of the range after each statement:

```
a.moveStart("word", -1)
a.moveEnd("sentence")
```

Notice that for a sentence, a default unit of 1 expands to the end of the current sentence. And if you move the start point backward by one sentence, you'll see that the lack of a period-ending sentence prior to the `myP` element causes strange results.

Finally, force the start point backward in increments of 20 words and watch the results as the starting point nears and reaches the start of the BODY:

```
a.moveStart("word", -20)
```

Eventually the last movement will be some other value closer to zero. And as soon as the range is at the beginning of the BODY element, the range can move no more in that direction, so the result is zero.

moveToBookmark("*bookmarkString*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 19-11 earlier in this chapter shows how to use the `moveToBookmark()` method to restore a text range so that changes that created the state saved by the bookmark can be undone. The `getBookmark()` method is used to save the snapshots, while the `moveToBookmark()` method is used during the undo process.

moveToElementText(*elemObjRef*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

A majority of examples for other `TextRange` object methods in this chapter use the `moveToElementText()` method. Listings 19-10 and 19-11 earlier in this chapter show the method within an application context.

moveToPoint(*x*, *y*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator to see the `moveToPoint()` method in action. Begin by creating a text range for the entire BODY element:

```
a = document.body.createTextRange()
```

Now, invoke the `moveToPoint()` method to a location 100, 100, which turns out to be in the rectangle space of the Results textarea:

```
a.moveToPoint(100,100)
```

If you type `a` into the bottom text box and view the properties, both the `htmlText` and `text` properties are empty because the insertion point represents no visible text content. But if you gradually move, for example, the start point backward one character at a time, you will see the `htmlText` and `text` properties begin to fill in with the body text that comes before the TEXTAREA element, namely the "Results:" label and the `<BR>` tag between it and the TEXTAREA element. Enter the following statement into the top text box and click the Evaluate button several times.

```
a.moveStart("character", -1)
```

Enter `a` into the bottom text box after each evaluation to list the properties of the range.

parentElement()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `parentElement()` method. Begin by setting the text range to the `myEM` element:

```
a = document.body.createTextRange()
a.moveToElementText(myEM)
```

To inspect the object returned by the `parentElement()` method, enter the following statement in the lower text box:

```
a.parentElement()
```

If you scroll down to the `outerHTML` property, you see that the parent of the text range is the `myEM` element, tag and all.

Next, extend the end point of the text range by one word:

```
a.moveEnd("word")
```

Because part of the text range now contains text of the `myP` object, the `outerHTML` property of `a.parentElement()` shows the entire `myP` element and tags.

pasteHTML("*HTMLText*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `pasteHTML()` method. The goal of the following sequence is to change the `<EM>` tag to a `<SPAN>` tag whose `STYLE` attribute sets the color of the original text that was in the EM element.

Begin by creating the text range and setting the boundaries to the `myEM` element:

```
a = document.body.createTextRange()
a.moveToElementText(myEM)
```

While you can pass the HTML string directly as a parameter to `pasteHTML()`, storing the HTML string in its own temporary variable may be more convenient (and more easily testable), such as:

```
b = "<SPAN STYLE='color:red'>" + a.text + "</SPAN>"
```

Notice that we concatenate the text of the current text range, because it has not yet been modified. Now we can paste the new HTML string into the current text range

```
a.pasteHTML(b)
```

At this point the EM element is gone from the object model, and the SPAN element is in its place. Prove it to yourself by looking at the HTML for the `myP` element:

```
myP.innerHTML
```

As noted earlier, the `pasteHTML()` method is not the only way to insert or replace HTML in a document. This method makes excellent sense when the user selects some text in the document to be replaced, because you can use the `document.selection.createRange()` method to get the text range for the selection. But if you're not using text ranges for other related operations, consider the other generic object properties and methods available to you.

select()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 19-11 earlier in this chapter for an example of the `select()` method in use.

setEndPoint("*type*", *otherRangeRef*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator to experiment with the `setEndPoint()` method. Begin by creating two independent text ranges, one for the `myP` element and one for `myEM`:

```
a = document.body.createTextRange()
a.moveToElementText(myP)
b = document.body.createTextRange()
b.moveToElementText(myEM)
```

Before moving any end points, compare the HTML for each of those ranges:

```
a.htmlText
b.htmlText
```

Now, move the start point of the `a` text range to the end point of the `b` text range:

```
a.setEndPoint("StartToEnd", b)
```

If you now view the HTML for the a range,

```
a.htmlText
```

you see that the `<P>` tag of the original `a` text range is nowhere to be found. This demonstration is a good lesson to use the `setEndPoint()` method primarily if you are concerned only with visible body text being inside ranges, rather than an element with its tags.

TextRectangle Object

Properties

bottom
left
right
top

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Listing 19-12 lets you click one of four nested elements to see how the `TextRectangle` is treated. When you click one of the elements, that element's `TextRectangle` dimension properties are used to set the size of a positioned element that highlights the space of the rectangle. Be careful not to confuse the visible rectangle object that you see on the page with the abstract `TextRectangle` object that is associated with each of the clicked elements.

An important part of the listing is the way the action of sizing and showing the positioned element is broken out as a separate function (`setHiliter()`) from the one that is the `onClick` event handler function (`handleClick()`). This is done so that the `onResize` event handler can trigger a script that gets the current rectangle for the last element clicked, and the positioned element can be sized and moved to maintain the highlight of the same text. As an experiment, try removing the `onResize` event handler from the `<BODY>` tag and watch what happens to the highlighted rectangle after you resize the browser window: the rectangle that represents the `TextRectangle` remains unchanged and loses track of the abstract `TextRectangle` associated with the actual element object.

Listing 19-12: **Using the TextRectangle Object Properties**

```
<HTML>
<HEAD>
<TITLE>TextRectangle Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// preserve reference to last clicked elem so resize can re-use it
var lastElem
// TextRectangle left tends to be out of registration by a couple of pixels
var rectLeftCorrection = 2
```

Continued

Listing 19-12 *(continued)*

```
// process mouse click
function handleClick() {
    var elem = event.srcElement
    if (elem.className && elem.className == "sample") {
        // set hiliter element only on a subset of elements
        lastElem = elem
        setHiliter()
    } else {
        // otherwise, hide the hiliter
        hideHiliter()
    }
}
function setHiliter() {
    if (lastElem) {
        var textRect = lastElem.getBoundingClientRect()
        hiliter.style.pixelTop = textRect.top + document.body.scrollTop
        hiliter.style.pixelLeft = textRect.left + document.body.scrollLeft -
            rectLeftCorrection
        hiliter.style.pixelHeight = textRect.bottom - textRect.top
        hiliter.style.pixelWidth = textRect.right - textRect.left
        hiliter.style.visibility = "visible"
    }
}
function hideHiliter() {
    hiliter.style.visibility = "hidden"
    lastElem = null
}
</SCRIPT>
</HEAD>
<BODY onClick="handleClick()" onResize="setHiliter()">
<H1>TextRectangle Object</H1>
<HR>
<P>Click on any of the four colored elements in the paragraph below and watch
the highlight rectangle adjust itself to the element's TextRectangle object.

<P CLASS="sample">Lorem ipsum dolor sit amet, <SPAN CLASS="sample"
STYLE="color:red">consectetaur adipisicing elit</SPAN>, sed do eiusmod tempor
<SPAN CLASS="sample" STYLE="color:green">incididunt ut labore et dolore <SPAN
CLASS="sample" STYLE="color:blue">magna aliqua</SPAN>. Ut enim adminim veniam,
quis nostrud exercitation ullamco</SPAN> laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit involuptate velit esse cillum
dolore eu fugiat nulla pariatur.</P>
<DIV ID="hiliter" STYLE="position:absolute; background-color:salmon; z-index:-1;
visibility:hidden"></DIV>
</BODY>
</HTML>
```

✦ ✦ ✦

CHAPTER 6

Image, Area, and Map Objects (Chapter 22)

The IMG element object is a popular scripting target, largely because it is easy to script it for effects such as mouse rollovers. Moreover, the element's scriptability extends backward in time to all but the very first generation of scriptable browsers. Playing a supporting role in image rollovers is the abstract `Image` object, which scripts use to pre-load images into the browser's cache for instantaneous image swapping. Even though the two objects manifest themselves differently within script operations, they share properties and methods, making it easy to learn their capabilities side by side.

AREA and MAP element objects work closely with each other. In practice, an AREA element resembles an A element that is set to work as a link. Both elements create clickable "hot spots" on the page that typically lead the user to other locations within the site or elsewhere on the Web. They also share a number of URL-related properties.

Examples Highlights

- Most IE browsers can load both still and motion images (such as MPEG movies) into an IMG element. Listing 22-3 shows how to swap between still and motion images via the `dynsrc` property.
- The page created from Listing 22-4 lets you compare the performance of swapping images with and without precaching. You also see how to have scripts rotate images on a timed schedule.
- Watch how the IMG element's `onLoad` event handler can trigger actions in Listing 22-5.
- A powerful Listing 22-7 demonstrates how scripts can fashion new client-side area maps when a different picture file loads into an IMG element.

Image and IMG Element Objects

Properties

align

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listing 22-1 enables you to choose from the different `align` property values as they influence the layout of an image whose HTML is embedded inline with some other text. Resize the window to see different perspectives on word-wrapping on a page and their effects on the alignment choices. Not all browsers provide distinctive alignments for each choice, so experiment in multiple supported browsers.

Listing 22-1: Testing an Image's align Property

```
<HTML>
<HEAD>
<TITLE>IMG align Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">

function setAlignment(sel) {
    document.myIMG.align = sel.options[sel.selectedIndex].text
}
</SCRIPT>
</HEAD>
<BODY>
<H1>IMG align Property</H1>
<HR>
<FORM>
Choose the image alignment:
<SELECT onChange="setAlignment(this)">
    <OPTION>absbottom
    <OPTION>absmiddle
    <OPTION>baseline
    <OPTION SELECTED >bottom
    <OPTION >left
    <OPTION>middle
    <OPTION>right
    <OPTION>texttop
    <OPTION>top
</SELECT>
</FORM>
```

```
<HR>
<P>Lorem ipsum dolor sit amet, consectetaur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. <IMG NAME="myIMG" SRC="desk1.gif" HEIGHT=90 WIDTH=120>
Ut enim adminim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.</P>
</BODY>
</HTML>
```

alt

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in *JavaScript Bible*) to assign a string to the `alt` property of the `document.myIMG` image on the page. First, assign a nonexistent image to the `src` property to remove the existing image:

```
document.myIMG.src = "fred.gif"
```

Scroll down to the image, and you can see a space for the image. Now, assign a string to the `alt` property:

```
document.myIMG.src = "Fred\'s face"
```

The extra backslash is required to escape the apostrophe inside the string. Scroll down to see the new `alt` text in the image space.

border

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

Feel free to experiment with the `document.myIMG.border` property for the image in The Evaluator (Chapter 13 in *JavaScript Bible*) by assigning different integer values to the property.

complete

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

To experiment with the `image.complete` property, quit and relaunch your browser before loading Listing 22-2 (in case the images are in memory cache). As each image loads, click the "Is it loaded yet?" button to see the status of the `complete` property for the `image` object. The value is `false` until the loading finishes; then, the value becomes `true`. The arch image is the bigger of the two image files. You may have to quit and relaunch your browser between trials to clear the arch image from the cache (or empty the browser's memory cache). If you experience difficulty with this property in your scripts, try adding an `onLoad` event handler (even if it is empty, as in Listing 22-2) to your `<IMG>` tag.

Listing 22-2: **Scripting image.complete**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript1.1">
function loadIt(theImage,form) {
    form.result.value = ""
    document.images[0].src = theImage
}
function checkLoad(form) {
    form.result.value = document.images[0].complete
}
</SCRIPT>
</HEAD>
<BODY>
<IMG SRC="cpu2.gif" WIDTH=120 HEIGHT=90 onLoad="">
<FORM>
<INPUT TYPE="button" VALUE="Load keyboard"
onClick="loadIt('cpu2.gif',this.form)">
<INPUT TYPE="button" VALUE="Load arch"
onClick="loadIt('arch.gif',this.form)"><P>
<INPUT TYPE="button" VALUE="Is it loaded yet?" onClick="checkLoad(this.form)">
<INPUT TYPE="text" NAME="result">
</FORM>
</BODY>
</HTML>
```

dynsrc

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

To swap between still and video sources, simply empty the opposite property. Listing 22-3 shows a simplified example that swaps between one fixed image and one video image. This listing exhibits most of the bugs associated with changing between static image and video sources described in the text.

Listing 22-3: **Changing Between Still and Motion Images**

```
<HTML>
<HEAD>
<TITLE>IMG dynsrc Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">

var trainImg = new Image(160,120)
trainImg.src = "amtrak.jpg"
trainImg.dynsrc = "amtrak.mpg"

function setLoop() {
    var selector = document.forms[0].looper
    document.myIMG.loop = selector.options[selector.selectedIndex].value
}

function setImage(type) {
    if (type == "jpg") {
        document.myIMG.dynsrc = ""
        document.myIMG.src = trainImg.src
    } else {
        document.myIMG.src = ""
        document.myIMG.start = "fileopen"
        setLoop()
        document.myIMG.dynsrc = trainImg.dynsrc
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>IMG dynsrc Property</H1>
<HR>
<FORM>
Choose image type:
<INPUT TYPE="radio" NAME="imgGroup" CHECKED onClick="setImage('jpg')">Still
```

Continued

Listing 22-3 *(continued)*

```
<INPUT TYPE="radio" NAME="imgGroup" onClick="setImage('mpg')">Video
<P>Play video how many times after loading:
<SELECT NAME="looper" onChange="setLoop()">
    <OPTION VALUE=1 SELECTED>Once
    <OPTION VALUE=2>Twice
    <OPTION VALUE=-1>Continuously
</SELECT></P>
</FORM>
<HR>
<IMG NAME="myIMG" SRC="amtrak.jpg" HEIGHT=120 WIDTH=160>
</BODY>
</HTML>
```

If you don't explicitly set the `start` property to `fileopen` (as shown in Listing 22-3), users of IE for the Macintosh have to double-click (IE4) or click (IE5) the movie image to make it run.

fileCreatedDate
fileModifiedDate
fileSize

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

These properties are similar to the same-named properties of the document object. You can see these properties in action in Listing 18-4. Make a copy of that listing, and supply an image before modifying the references from the document object to the image object to see how these properties work with the IMG element object.

height
width

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in *JavaScript Bible*) to experiment with the `height` and `width` properties. Begin retrieving the default values by entering the following two statements into the top text box:

```
document.myIMG.height
document.myIMG.width
```

Increase the height of the image from its default 90 to 180:

```
document.myIMG.height = 180
```

If you scroll down to the image, you see that the image has scaled in proportion. Next, exaggerate the width:

```
document.myIMG.width = 400
```

View the resulting image.

hspace
vspace

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in *JavaScript Bible*) to experiment with the `hspace` and `vspace` properties. Begin by noticing that the image near the bottom of the page has no margins specified for it and is flush left with the page. Now assign a horizontal margin spacing of 30 pixels:

```
document.myIMG.hspace = 30
```

The image has shifted to the right by 30 pixels. An invisible margin also exists to the right of the image.

isMap

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The image in The Evaluator page is not defined as an image map. Thus, if you type the following statement into the top text box, the property returns `false`:

```
document.myIMG.isMap
```

loop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 22-3 for the `dynsrc` property to see the `loop` property in action.

lowsrc
lowSrc

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

See Listing 22-5 for the image object's `onLoad` event handler to see how the source-related properties affect event processing.

name

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	(✓)		✓	✓	✓

Example

You can use The Evaluator to examine the value returned by the `name` property of the image on that page. Enter the following statement into the top text box:

```
document.myIMG.name
```

Of course, this is redundant because the name is part of the reference to the object.

nameProp

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

You can use The Evaluator to compare the results of the `src` and `nameProp` properties in IE5+/Windows. Enter each of the following statements into the top text box:

```
document.myIMG.src
document.myIMG.nameProp
```

protocol

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

You can use The Evaluator to examine the `protocol` property of the image on the page. Enter the following statement into the top text box:

```
document.myIMG.protocol
```

src

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓	(✓)		✓	✓	✓

Example

In the following example (Listing 22-4), you see a few applications of image objects. Of prime importance is a comparison of how precached and regular images feel to the user. As a bonus, you see an example of how to set a timer to automatically change the images displayed in an image object. This feature is a popular request among sites that display advertising banners.

As the page loads, a global variable is handed an array of image objects. Entries of the array are assigned string names as index values (`"desk1"`, `"desk2"`, and so on). The intention is that these names ultimately will be used as addresses to the array entries. Each image object in the array has a URL assigned to it, which precaches the image.

The page (see Figure 6-1) includes two IMG elements: one that displays non-cached images and one that displays cached images. Under each image is a SELECT element that you can use to select one of four possible image files for each element. The `onChange` event handler for each SELECT list invokes a different function to change the noncached (`loadIndividual()`) or cached (`loadCached()`) images. Both of these functions take as their single parameter a reference to the form that contains the SELECT elements.

To cycle through images at five-second intervals, the `checkTimer()` function looks to see if the timer check box is checked. If so, the `selectedIndex` property of the cached image SELECT control is copied and incremented (or reset to zero if the index is at the maximum value). The SELECT element is adjusted, so you can now invoke the `loadCached()` function to read the currently selected item and set the image accordingly.

For some extra style points, the `<BODY>` tag includes an `onUnload` event handler that invokes the `resetSelects()` function. This general-purpose function loops through all forms on the page and all elements within each form. For every SELECT element, the `selectedIndex` property is reset to zero. Thus, if a user reloads the page, or returns to the page via the Back button, the images start in their original sequence. An `onLoad` event handler makes sure that the images are in sync with the SELECT choices and the `checkTimer()` function is invoked with a five-second delay. Unless the timer check box is checked, however, the cached images don't cycle.

Listing 22-4: **A Scripted Image Object and Rotating Images**

```
<HTML>
<HEAD>
<TITLE>Image Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// global declaration for 'desk' images array
var imageDB
// pre-cache the 'desk' images
if (document.images) {
    // list array index names for convenience
    var deskImages = new Array("desk1", "desk2", "desk3", "desk4")
    // build image array and pre-cache them
    imageDB = new Array(4)
    for (var i = 0; i < imageDB.length ; i++) {
        imageDB[deskImages[i]] = new Image(120,90)
        imageDB[deskImages[i]].src = deskImages[i] + ".gif"
    }
}
// change image of 'individual' image
function loadIndividual(form) {
    if (document.images) {
        var gifName = form.individual.options[form.individual.selectedIndex].value
        document.thumbnail1.src = gifName + ".gif"
    }
}
// change image of 'cached' image
function loadCached(form) {
    if (document.images) {
```

```
        var gifIndex = form.cached.options[form.cached.selectedIndex].value
        document.thumbnail2.src = imageDB[gifIndex].src
    }
}
// if switched on, cycle 'cached' image to next in queue
function checkTimer() {
    if (document.images && document.Timer.timerBox.checked) {
        var gifIndex = document.selections.cached.selectedIndex
        if (++gifIndex > imageDB.length - 1) {
            gifIndex = 0
        }
        document.selections.cached.selectedIndex = gifIndex
        loadCached(document.selections)
        var timeoutID = setTimeout("checkTimer()",5000)
    }
}
// reset form controls to defaults on unload
function resetSelects() {
    for (var i = 0; i < document.forms.length; i++) {
        for (var j = 0; j < document.forms[i].elements.length; j++) {
            if (document.forms[i].elements[j].type == "select-one") {
                document.forms[i].elements[j].selectedIndex = 0
            }
        }
    }
}
// get things rolling
function init() {
    loadIndividual(document.selections)
    loadCached(document.selections)
    setTimeout("checkTimer()",5000)
}
</SCRIPT>
</HEAD>

<BODY onLoad="init()" onUnload="resetSelects ()">
<H1>Image Object</H1>
<HR>
<CENTER>
<TABLE BORDER=3 CELLPADDING=3>
<TR><TH></TH><TH>Individually Loaded</TH><TH>Pre-cached</TH></TR>
<TR><TD ALIGN=RIGHT><B>Image:</B></TD>
<TD><IMG SRC="cpu1.gif" NAME="thumbnail1" HEIGHT=90 WIDTH=120></TD>
<TD><IMG SRC="desk1.gif" NAME="thumbnail2" HEIGHT=90 WIDTH=120></TD>
</TR>
<TR><TD ALIGN=RIGHT><B>Select image:</B></TD>
<FORM NAME="selections">
<TD>
<SELECT NAME="individual" onChange="loadIndividual(this.form)">
<OPTION VALUE="cpu1">Wires
<OPTION VALUE="cpu2">Keyboard
```

Continued

Listing 22-4 *(continued)*

```
<OPTION VALUE="cpu3">Desks
<OPTION VALUE="cpu4">Cables
</SELECT>
</TD>
<TD>
<SELECT NAME="cached" onChange="loadCached(this.form)">
<OPTION VALUE="desk1">Bands
<OPTION VALUE="desk2">Clips
<OPTION VALUE="desk3">Lamp
<OPTION VALUE="desk4">Erasers
</SELECT></TD>
</FORM>
</TR></TABLE>
<FORM NAME="Timer">
<INPUT TYPE="checkbox" NAME="timerBox" onClick="checkTimer()">Auto-cycle through
pre-cached images
</FORM>
</CENTER>
</BODY>
</HTML>
```

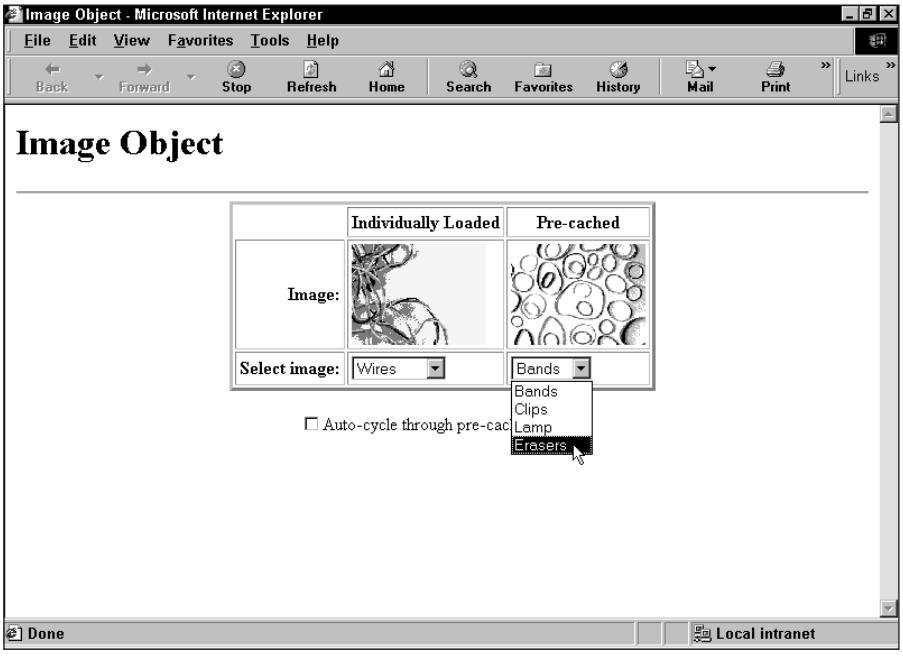


Figure 6-1: The image object demonstration page (Images © Aris Multimedia Entertainment, Inc., 1994)

start

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 22-3 earlier in this chapter for an example of how you can use the `start` property with a page that loads a movie clip into an IMG element object.

x
y

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

If you want to scroll the document so that the link is a few pixels below the top of the window, use a statement such as this:

```
window.scrollTo(document.images[0].x, (document.images[0].y - 3))
```

Event handlers

onAbort
onError

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

Listing 22-5 includes an `onAbort` event handler. If the images already exist in the cache, you must quit and relaunch the browser to try to stop the image from loading. In that example, I provide a reload option for the entire page. How you handle the exception depends a great deal on your page design. Do your best to smooth over any difficulties that users may encounter.

onLoad

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

Quit and restart your browser to get the most from Listing 22-5. As the document first loads, the LOWSRC image file (the picture of pencil erasers) loads ahead of the computer keyboard image. When the erasers are loaded, the `onLoad` event handler writes "done" to the text field even though the main image is not loaded yet. You can experiment further by loading the arch image. This image takes longer to load, so the LOWSRC image (set on the fly, in this case) loads way ahead of it.

Listing 22-5: **The Image onLoad Event Handler**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function loadIt(theImage,form) {
    if (document.images) {
        form.result.value = ""
        document.images[0].lowsrc = "desk1.gif"
        document.images[0].src = theImage
    }
}
function checkLoad(form) {
    if (document.images) {
        form.result.value = document.images[0].complete
    }
}
function signal() {
    if(confirm("You have stopped the image from loading. Do you want to try
again?")) {
        location.reload()
    }
}
</SCRIPT>
</HEAD>
<BODY>
<IMG SRC="cpu2.gif" LOWSRC="desk4.gif" WIDTH=120 HEIGHT=90
onLoad="if (document.forms[0].result) document.forms[0].result.value='done'"
onAbort="signal()">
<FORM>
<INPUT TYPE="button" VALUE="Load keyboard"
onClick="loadIt('cpu2.gif',this.form)">
<INPUT TYPE="button" VALUE="Load arch"
onClick="loadIt('arch.gif',this.form)"><P>
```

```
<INPUT TYPE="button" VALUE="Is it loaded yet?" onClick="checkLoad(this.form)">
<INPUT TYPE="text" NAME="result">
</FORM>
</BODY>
</HTML>
```

AREA Element Object

Properties

coords
shape

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See Listing 22-7 for a demonstration of the coords and shape properties in the context of scripting MAP element objects.

MAP Element Object

Property

areas

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listing 22-7 demonstrates how to use scripting to replace the AREA element objects inside a MAP element. The scenario is that the page loads with one image of a computer keyboard. This image is linked to the keyboardMap client-side image map, which specifies details for three hot spots on the image. If you then switch the

image displayed in that IMG element, scripts change the `useMap` property of the IMG element object to point to a second MAP that has specifications more suited to the desk lamp in the second image. Roll the mouse pointer atop the images, and view the URLs associated with each area in the statusbar (for this example, the URLs do not lead to other pages).

Another button on the page, however, invokes the `makeAreas()` function (not working in IE5/Mac), which creates four new AREA element objects and (through DOM-specific pathways) adds those new area specifications to the image. If you roll the mouse atop the image after the function executes, you can see that the URLs now reflect those of the new areas. Also note the addition of a fourth area, whose status bar message appears in Figure 6-2.

Listing 22-7: **Modifying AREA Elements on the Fly**

```
<HTML>
<HEAD>
<TITLE>MAP Element Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// generate area elements on the fly
function makeAreas() {
    document.myIMG.src = "desk3.gif"
    // build area element objects
    var area1 = document.createElement("AREA")
    area1.href = "Script-Made-Shade.html"
    area1.shape = "polygon"
    area1.coords = "52,28,108,35,119,29,119,8,63,0,52,28"
    var area2 = document.createElement("AREA")
    area2.href = "Script-Made-Base.html"
    area2.shape = "rect"
    area2.coords = "75,65,117,87"
    var area3 = document.createElement("AREA")
    area3.href = "Script-Made-Chain.html"
    area3.shape = "polygon"
    area3.coords = "68,51,73,51,69,32,68,51"
    var area4 = document.createElement("AREA")
    area4.href = "Script-Made-Emptyness.html"
    area4.shape = "rect"
    area4.coords = "0,0,50,120"
    // stuff new elements into MAP child nodes
    if (document.all) {
        // works for IE4+
        document.all.lampMap.areas.length = 0
        document.all.lampMap.areas[0] = area1
        document.all.lampMap.areas[1] = area2
        document.all.lampMap.areas[2] = area3
        document.all.lampMap.areas[3] = area4
```

```
    } else if (document.getElementById) {
        // NN6 adheres to node model
        var mapObj = document.getElementById("lamp_map")
        while (mapObj.childNodes.length) {
            mapObj.removeChild(mapObj.firstChild)
        }
        mapObj.appendChild(area1)
        mapObj.appendChild(area2)
        mapObj.appendChild(area3)
        mapObj.appendChild(area4)
        // workaround NN6 display bug
        document.myIMG.style.display = "inline"
    }
}

function changeToKeyboard() {
    document.myIMG.src = "cpu2.gif"
    document.myIMG.useMap = "#keyboardMap"
}

function changeToLamp() {
    document.myIMG.src = "desk3.gif"
    document.myIMG.useMap = "#lampMap"
}
</SCRIPT>
</HEAD>
<BODY>
<H1>MAP Element Object</H1>
<HR>
<IMG NAME="myIMG" SRC="cpu2.gif" WIDTH=120 HEIGHT=90 USEMAP="#keyboardMap">
<FORM>
<P><INPUT TYPE="button" VALUE="Load Lamp Image" onClick="changeToLamp()">
<INPUT TYPE="button" VALUE="Write Map on the Fly" onClick="makeAreas()"></P>
<P>
<INPUT TYPE="button" VALUE="Load Keyboard Image"
onClick="changeToKeyboard()"></P>
</FORM>
<MAP NAME="keyboardMap">
<AREA HREF="AlpaKeys.htm" SHAPE="rect" COORDS="0,0,26,42">
<AREA HREF="ArrowKeys.htm" SHAPE="polygon"
COORDS="48,89,57,77,69,82,77,70,89,78,84,89,48,89">
<AREA HREF="PageKeys.htm" SHAPE="circle" COORDS="104,51,14">
</MAP>
<MAP NAME="lampMap">
<AREA HREF="Shade.htm" SHAPE="polygon"
COORDS="52,28,108,35,119,29,119,8,63,0,52,28">
<AREA HREF="Base.htm" SHAPE="rect" COORDS="75,65,117,87">
<AREA HREF="Chain.htm" SHAPE="polygon" COORDS="68,51,73,51,69,32,68,51">
</MAP>
</BODY>
</HTML>
```

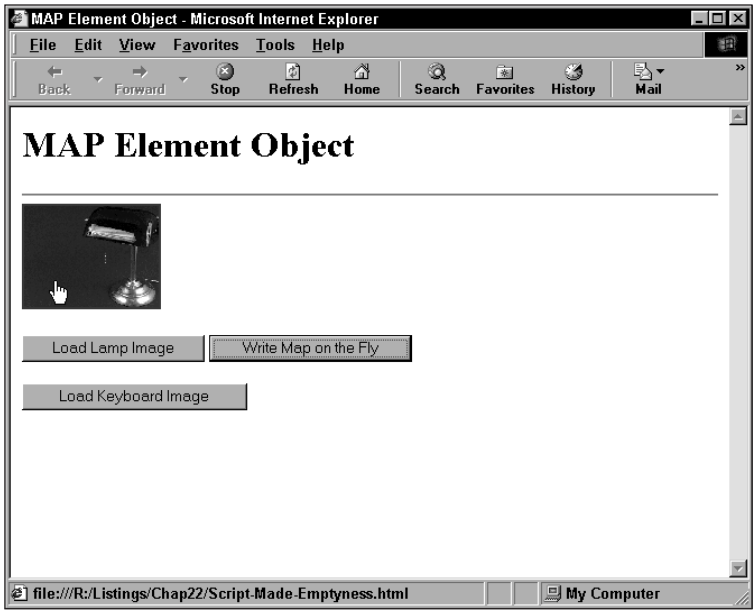


Figure 6-2: Scripts created a special client-side image map for the image.

✦ ✦ ✦

CHAPTER 7

The Form and Related Objects (Chapter 23)

In This Chapter

Customizing FORM object behavior prior to submission

Preventing accidental form submissions or resets

Using images for Reset and Submit buttons

Processing form validations

Because HTML forms have been scriptable since the earliest days of scriptable browsers, they tend to attract the attention of a lot of page and site designers. Even though the FORM element is primarily the container of the interactive form controls (covered in succeeding chapters), it's not uncommon to find scripts modifying the `action` property (corresponding to the ACTION attribute) based on user input. Moreover, the `onSubmit` event handler is a vital trigger for batch validation just before the form data goes up to the server.

The other HTML element for which this chapter contains an example is the LABEL element object. A LABEL element is a container of text that is associated with a form control. This is a practical user interface enhancement in modern browsers in that such labels can essentially forward mouse events to their controls, thus widening the physical target for mouse clicks of radio buttons and checkboxes, much like "real" applications. The value of scriptability for this element, however, accrues predominantly when scripts dynamically modify page content.

Examples Highlights

- Listing 23-2 puts the `form.elements` array to work in a generic function that resets all text fields in a form to empty, without touching the settings of other types of controls.
- If you prefer to use images for your form's reset and submit actions, Listing 22-3 shows you how to do just that with the `form.reset()` and `form.submit()` methods.

✦ While batch form validations are shown in several places throughout the *JavaScript Bible*, Listing 23-4 demonstrates how both the `onReset` and `onSubmit` event handlers, in concert with the `window.confirm()` method, let scripts permit or prevent a form from being reset or submitted.

FORM Object

Properties

action

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The following statement assigns a `mailto:` URL to the first form of a page:

```
document.forms[0].action = "mailto:jdoe@giantco.com"
```

elements

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The document in Listing 23-2 demonstrates a practical use of the `elements` property. A form contains four fields and some other elements mixed in between (see Figure 7-1). The first part of the function that acts on these items repeats through all the elements in the form to find out which ones are text box objects and which text box objects are empty. Notice how I use the `type` property to separate text box objects from the rest, even when radio buttons appear amid the fields. If one field has nothing in it, I alert the user and use that same index value to place the insertion point at the field with the field's `focus()` method.

Listing 23-2: **Using the form.elements Array**

```
<HTML>
<HEAD>
<TITLE>Elements Array</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function verifyIt() {
    var form = document.forms[0]
    for (i = 0; i < form.elements.length; i++) {
        if (form.elements[i].type == "text" && form.elements[i].value == ""){
            alert("Please fill out all fields.")
            form.elements[i].focus()
            break
        }
        // more tests
    }
    // more statements
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
Enter your first name:<INPUT TYPE="text" NAME="firstName"><P>
Enter your last name:<INPUT TYPE="text" NAME="lastName"><P>
<INPUT TYPE="radio" NAME="gender">Male
<INPUT TYPE="radio" NAME="gender">Female <P>
Enter your address:<INPUT TYPE="text" NAME="address"><P>
Enter your city:<INPUT TYPE="text" NAME="city"><P>
<INPUT TYPE="checkbox" NAME="retired">I am retired
</FORM>
<FORM>
<INPUT TYPE="button" NAME="act" VALUE="Verify" onClick="verifyIt()">
</FORM>
</BODY>
</HTML>
```

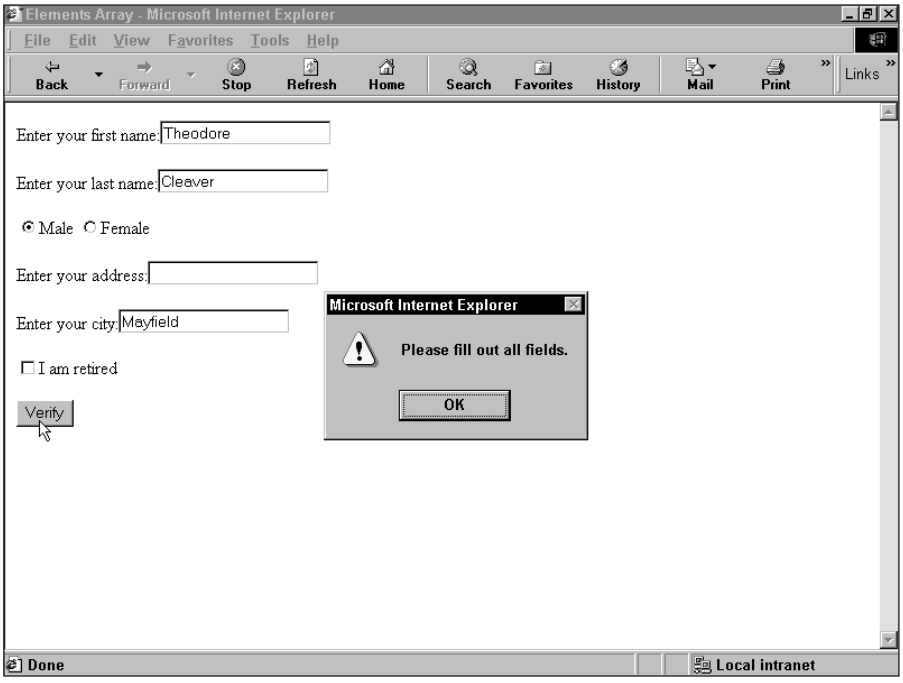


Figure 7-1: The elements array helps find text fields for validation.

encoding
enctype

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

If you need to modify the first form in a document so that the content is sent in non-URL-encoded text at the user's request, the statement is:

```
document.forms[0].encoding = "text/plain"
```

length

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to determine the number of form controls in the first form of the page. Enter the following statement into the top text box:

```
document.forms[0].length
```

method

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

If you need to modify the first form in a document so that the content is sent via the `POST` method, the statement is:

```
document.forms[0].method = "POST"
```

target

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

If you want to direct the response from the first form's CGI to a new window (rather than the target specified in the form's tag), use this statement:

```
document.forms[0].target = "_blank"
```

Methods

reset()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

In Listing 23-3, I assign the act of resetting the form to the `HREF` attribute of a link object (that is attached to a graphic called `reset.jpg`). I use the `javascript:` URL to invoke the `reset()` method for the form directly (in other words, without doing it via function). Note that the form's action in this example is to a nonexistent URL. If you click the Submit icon, you receive an "unable to locate" error from the browser.

Listing 23-3: form.reset() and form.submit() Methods

```
<HTML>
<HEAD>
<TITLE>Registration Form</TITLE>
</HEAD>
<BODY>
<FORM NAME="entries" METHOD=POST ACTION="http://www.u.edu/pub/cgi-bin/register">
Enter your first name:<INPUT TYPE="text" NAME="firstName"><P>
Enter your last name:<INPUT TYPE="text" NAME="lastName"><P>
Enter your address:<INPUT TYPE="text" NAME="address"><P>
Enter your city:<INPUT TYPE="text" NAME="city"><P>
<INPUT TYPE="radio" NAME="gender" CHECKED>Male
<INPUT TYPE="radio" NAME="gender">Female <P>
<INPUT TYPE="checkbox" NAME="retired">I am retired
</FORM>
<P>
<A HREF="javascript:document.forms[0].submit()"><IMG SRC="submit.jpg" HEIGHT=25
WIDTH=100 BORDER=0></A>
<A HREF="javascript:document.forms[0].reset()"><IMG SRC="reset.jpg" HEIGHT=25
WIDTH=100 BORDER=0></A>
</BODY>
</HTML>
```

submit()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Consult Listing 23-3 for an example of using the `submit()` method from outside of a form.

Event handlers

onReset

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

Listing 23-4 demonstrates one way to prevent accidental form resets or submissions. Using standard Reset and Submit buttons as interface elements, the `<FORM>` object definition includes both event handlers. Each event handler calls its own function that offers a choice for users. Notice how each event handler includes the word `return` and takes advantage of the Boolean values that come back from the `confirm()` method dialog boxes in both functions.

Listing 23-4: The onReset and onSubmit Event Handlers

```
<HTML>
<HEAD>
<TITLE>Submit and Reset Confirmation</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function allowReset() {
    return window.confirm("Go ahead and clear the form?")
}
function allowSend() {
    return window.confirm("Go ahead and mail this info?")
}
</SCRIPT>
</HEAD>
<BODY>
<FORM METHOD=POST ENCTYPE="text/plain" ACTION="mailto:trash4@dannyg.com"
onReset="return allowReset()" onSubmit="return allowSend()">
```

Continued

Listing 23-4 *(continued)*

```
Enter your first name:<INPUT TYPE="text" NAME="firstName"><P>
Enter your last name:<INPUT TYPE="text" NAME="lastName"><P>
Enter your address:<INPUT TYPE="text" NAME="address"><P>
Enter your city:<INPUT TYPE="text" NAME="city"><P>
<INPUT TYPE="radio" NAME="gender" CHECKED>Male
<INPUT TYPE="radio" NAME="gender">Female <P>
<INPUT TYPE="checkbox" NAME="retired">I am retired<P>
<INPUT TYPE="reset">
<INPUT TYPE="submit">
</FORM>
</BODY>
</HTML>
```

onSubmit

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listing 23-4 for an example of trapping a submission via the `onSubmit` event handler.

LABEL Element Object

Property

htmlFor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statement uses W3C DOM-compatible syntax (IE5+ and NN6) to assign a form control reference to the `htmlFor` property of a label:

```
document.getElementById("myLabel").htmlFor = document.getElementById("myField")
```

✦ ✦ ✦

CHAPTER 8

Button Objects (Chapter 24)

In This Chapter

Triggering action from a user's click of a button

Using checkboxes to control display of other form controls

Distinguishing between radio button families and their individual buttons

The topic of button form controls encompasses clickable user interface elements that have a variety of applications, some of which are quite specific. For example, radio buttons should be presented in groups offering two or more mutually exclusive choices. A checkbox, on the other hand, is used to signify an "on" or "off" setting related to whatever label is associated with the button. The only tricky part of these special behaviors is that radio buttons assigned to a single group must share the same name, and the document object model provides access to single buttons within the group by way of an array of objects that share the name. For a script to determine which radio button is currently selected, a `for` loop through the array then allows the script to inspect the `checked` property of each button to find the one whose value is `true`.

Then there are what appear to be plain old rounded rectangle buttons. Two versions — the INPUT element of type button and the newer BUTTON element — work very much alike, although the latter is not obligated to appear nested inside a FORM element. A common mistake among newcomers, however, is to use the INPUT element of type `submit` to behave as a button whose sole job is to trigger some script function without any form submission. Genuine submit buttons force the form to submit itself, even if the button's `onClick` event handler invokes a script function. If the form has no `ACTION` attribute assigned to it, then the default action of the submission causes the page to reload, probably destroying whatever tentative script variable values and other data have been gathered on the page.

Examples Highlights

- If a button's event handler passes that button object's reference to the handler function, the object's `form` property provides the function with a valid reference to the containing form, allowing the script an easy way to access information about the form or create references to other form controls.

- ✦ Of course, the `onClick` event handler is the most important for button controls. Listing 24-1 demonstrates passing button references to event handler functions.
- ✦ Listing 24-4 shows how a checkbox setting can influence the URL of the form's action.
- ✦ Sometimes a complex form requires that checking a checkbox makes other items in the form visible. Listing 24-5 employs scriptable style sheets to assist in the job.
- ✦ Use Listing 24-6 as a model for how to find which radio button among those of a single group is checked.

The BUTTON Element Object and the Button, Submit, and Reset Input Objects

Properties

form

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The following function fragment receives a reference to a button element as the parameter. The button reference is needed to decide which branch to follow; then the form is submitted.

```
function setAction(btn) {
    if (btn.name == "normal") {
        btn.form.action = "cgi-bin/normal.pl"
    } else if (btn.name == "special") {
        btn.form.action = "cgi-bin/specialHandling.pl"
    }
    btn.form.submit()
}
```

Notice how this function doesn't have to worry about the form reference, because its job is to work with whatever form encloses the button that triggers this function. Down in the form, two buttons invoke the same function. Only their names ultimately determine the precise processing of the button click:

```
<FORM>
...
<INPUT TYPE="button" NAME="normal" VALUE="Regular Handling"
onClick="setAction(this)">
```

```
<INPUT TYPE="button" NAME="special" VALUE="Special Handling"
onClick="setAction(this)">
</FORM>
```

name

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See the example for the `form` property earlier in this chapter for a practical application of the `name` property.

value

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

In the following excerpt, the statement toggles the label of a button from "Play" to "Stop" (except in NN/Mac through version 4):

```
var btn = document.forms[0].controlButton
btn.value = (btn.value == "Play") ? "Stop" : "Play"
```

Methods

click()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The following statement demonstrates how to script a click action on a button form control named `sender`:

```
document.forms[0].sender.click()
```

Event handlers

onClick

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 24-1 demonstrates not only the `onClick` event handler of a button but also how you may need to extract a particular button's `name` or `value` properties from a general-purpose function that services multiple buttons. In this case, each button passes its own object as a parameter to the `displayTeam()` function. The function then displays the results in an alert dialog box. A real-world application would probably use a more complex `if...else` decision tree to perform more sophisticated actions based on the button clicked (or use a `switch` construction on the `btn.value` expression for NN4+ and IE4+).

Listing 24-1: **Three Buttons Sharing One Function**

```
<HTML>
<HEAD>
<TITLE>Button Click</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function displayTeam(btn) {
    if (btn.value == "Abbott") {alert("Abbott & Costello")}
    if (btn.value == "Rowan") {alert("Rowan & Martin")}
    if (btn.value == "Martin") {alert("Martin & Lewis")}
}
</SCRIPT>
</HEAD>

<BODY>
Click on your favorite half of a popular comedy team:<P>
<FORM>
<INPUT TYPE="button" VALUE="Abbott" onClick="displayTeam(this)">
<INPUT TYPE="button" VALUE="Rowan" onClick="displayTeam(this)">
<INPUT TYPE="button" VALUE="Martin" onClick="displayTeam(this)">
</FORM>
</BODY>
</HTML>
```

Checkbox Input Object

Properties

checked

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The simple example in Listing 24-2 passes a form object reference to the JavaScript function. The function, in turn, reads the `checked` value of the form's checkbox object (`checkThis.checked`) and uses its Boolean value as the test result for the `if...else` construction.

Listing 24-2: The checked Property as a Conditional

```
<HTML>
<HEAD>
<TITLE>Checkbox Inspector</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function inspectBox(form) {
    if (form.checkThis.checked) {
        alert("The box is checked.")
    } else {
        alert("The box is not checked at the moment.")
    }
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="checkbox" NAME="checkThis">Check here<P>
<INPUT TYPE="button" NAME="boxChecker" VALUE="Inspect Box"
onClick="inspectBox(this.form)">
</FORM>
</BODY>
</HTML>
```

defaultChecked

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The function in Listing 24-3 (this fragment is not in the CD-ROM listings) is designed to compare the current setting of a checkbox against its default value. The `if` construction compares the current status of the box against its default status. Both are Boolean values, so they can be compared against each other. If the current and default settings don't match, the function goes on to handle the case in which the current setting is other than the default.

Listing 24-3: Examining the defaultChecked Property

```
function compareBrowser(thisBox) {
    if (thisBox.checked != thisBox.defaultChecked) {
        // statements about using a different set of HTML pages
    }
}
```

value

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The scenario for the skeleton HTML page in Listing 24-4 is a form with a checkbox whose selection determines which of two actions to follow for submission to the server. After the user clicks the Submit button, a JavaScript function examines the checkbox's `checked` property. If the property is `true` (the button is checked), the script sets the `action` property for the entire form to the content of the value property — thus influencing where the form goes on the server side. If you try this listing on your computer, the result you see varies widely with the browser version you use. For most browsers, you see some indication (an error alert or other screen notation) that a file with the name `primaryURL` or `alternateURL` doesn't exist. Unfortunately, IE5.5/Windows does not display the name of the file that can't be opened. Try the example in another browser if you have one. The names and the error message come from the submission process for this demonstration.

Listing 24-4: **Adjusting a CGI Submission Action**

```
<HTML>
<HEAD>
<TITLE>Checkbox Submission</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setAction(form) {
    if (form.checkThis.checked) {
        form.action = form.checkThis.value
    } else {
        form.action = "file://primaryURL"
    }
    return true
}
</SCRIPT>
</HEAD>
<BODY>
<FORM METHOD="POST" ACTION="">
<INPUT TYPE="checkbox" NAME="checkThis" VALUE="file://alternateURL">Use
alternate<P>
<INPUT TYPE="submit" NAME="boxChecker" onClick="return setAction(this.form)">
</FORM>
</BODY>
</HTML>
```

Event handlers

onClick

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The page in Listing 24-5 shows how to trap the click event in one checkbox to influence the visibility and display of other form controls. After you turn on the Monitor checkbox, a list of radio buttons for monitor sizes appears. Similarly, engaging the Communications checkbox makes two radio buttons visible. Your choice of radio button brings up one of two further choices within the same table cell (see Figure 8-1).

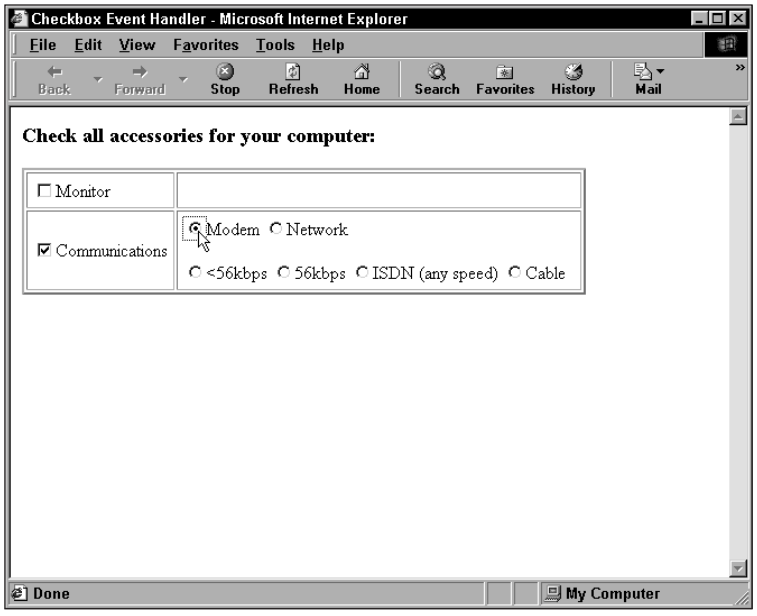


Figure 8-1: Clicking on button choices reveals additional relevant choices

Notice how the `toggle()` function was written as a generalizable function. This function can accept a reference to any checkbox object and any related span. If five more groups like this were added to the table, no additional functions would be needed.

In the `swap()` function, an application of a nested `if...else` shortcut construction is used to convert the Boolean values of the `checked` property to the strings needed for the `display` style property. The nesting is used to allow a single statement to take care of two conditions: the group of buttons to be controlled and the `checked` property of the button invoking the function. This function is not generalizable, because it contains explicit references to objects in the document. The `swap()` function can be made generalizable, but due to the special relationships between pairs of span elements (meaning one has to be hidden while the other displayed in its place), the function would require more parameters to fill in the blanks where explicit references are needed.

A rendering bug in NN6 causes the form controls in the lower right frame to lose their settings when the elements have their `display` style property set to `none`. The problem is related to the inclusion of P or similar block elements inside a table cell that contains controls. Therefore, if you uncheck and recheck the Communications checkbox in the example page, the previously displayed subgroup shows up even though no radio buttons are selected. You can script around this bug by preserving radio button settings in a global variable as you hide the group, and restoring the settings when you show the group again.

Syntax used to address elements here is the W3C DOM-compatible form, so this listing runs as is with IE5+ and NN6+. You can modify the listing to run in IE4 by adapting references to the `document.all` format.

Listing 24-5: **A Checkbox and an onClick event Handler**

```
<HTML>
<HEAD>
<TITLE>Checkbox Event Handler</TITLE>
<STYLE TYPE="text/css">
#monGroup {visibility:hidden}
#comGroup {visibility:hidden}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
// toggle visibility of a main group spans
function toggle(chkbox, group) {
    var visSetting = (chkbox.checked) ? "visible" : "hidden"
    document.getElementById(group).style.visibility = visSetting
}
// swap display of communications sub group spans
function swap(radBtn, group) {
    var modemsVisSetting = (group == "modems") ?
        ((radBtn.checked) ? "" : "none") : "none"
    var netwksVisSetting = (group == "netwks") ?
        ((radBtn.checked) ? "" : "none") : "none"
    document.getElementById("modems").style.display = modemsVisSetting
    document.getElementById("netwks").style.display = netwksVisSetting
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<H3>Check all accessories for your computer:</H3>
<TABLE BORDER=2 CELLPADDING=5>
<TR>
    <TD>
    <INPUT TYPE="checkbox" NAME="monitor"
    onClick="toggle(this, 'monGroup')">Monitor
    </TD>
    <TD>
    <SPAN ID="monGroup">
        <INPUT TYPE="radio" NAME="monitorType">15"
        <INPUT TYPE="radio" NAME="monitorType">17"
        <INPUT TYPE="radio" NAME="monitorType">21"
        <INPUT TYPE="radio" NAME="monitorType">>21"
    </SPAN>
    </TD>
</TR>
<TR>
    <TD>
```

Continued

Listing 24-5 *(continued)*

```
    <INPUT TYPE="checkbox" NAME="comms"
        onClick="toggle(this, 'comGroup')">Communications
    </TD>
    <TD>
    <SPAN ID="comGroup">
        <P><INPUT TYPE="radio" NAME="commType"
            onClick="swap(this, 'modems')">Modem
        <INPUT TYPE="radio" NAME="commType"
            onClick="swap(this, 'netwks')">Network</P>
            <P><SPAN ID="modems" STYLE="display:none">
                <INPUT TYPE="radio" NAME="modemType"><56kbps
                <INPUT TYPE="radio" NAME="modemType">56kbps
                <INPUT TYPE="radio" NAME="modemType">ISDN (any speed)
                <INPUT TYPE="radio" NAME="modemType">Cable
            </SPAN>
            <SPAN ID="netwks" STYLE="display:none">
                <INPUT TYPE="radio" NAME="netwkType">Ethernet 10Mbps (10-Base T)
                <INPUT TYPE="radio" NAME="netwkType">Ethernet 100Mbps (10/100)
                <INPUT TYPE="radio" NAME="netwkType">T1 or greater
            </SPAN> </P>
    </SPAN>
    </TD>
</TR>

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

Radio Input Object

Properties

checked

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 24-6 uses a repeat loop in a function to look through all buttons in the Stooges group in search of the checked button. After the loop finds the one whose

checked property is `true`, it returns the value of the index. In one instance, that index value is used to extract the `value` property for display in the alert dialog box; in the other instance, the value helps determine which button in the group is next in line to have its checked property set to `true`.

Listing 24-6: **Finding the Selected Button in a Radio Group**

```
<HTML>
<HEAD>
<TITLE>Extracting Highlighted Radio Button</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function getSelectedButton(buttonGroup){
    for (var i = 0; i < buttonGroup.length; i++) {
        if (buttonGroup[i].checked) {
            return i
        }
    }
    return 0
}
function fullName(form) {
    var i = getSelectedButton(form.stooges)
    alert("You chose " + form.stooges[i].value + ".")
}
function cycle(form) {
    var i = getSelectedButton(form.stooges)
    if (i+1 == form.stooges.length) {
        form.stooges[0].checked = true
    } else {
        form.stooges[i+1].checked = true
    }
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<B>Select your favorite Stooge:</B>
<P><INPUT TYPE="radio" NAME="stooges" VALUE="Moe Howard" CHECKED>Moe
<INPUT TYPE="radio" NAME="stooges" VALUE="Larry Fine" >Larry
<INPUT TYPE="radio" NAME="stooges" VALUE="Curly Howard" >Curly
<INPUT TYPE="radio" NAME="stooges" VALUE="Shemp Howard" >Shemp</P>
<P><INPUT TYPE="button" NAME="Viewer" VALUE="View Full Name..."
onClick="fullName(this.form)"></P>
<P><INPUT TYPE="button" NAME="Cycler" VALUE="Cycle Buttons"
onClick="cycle(this.form)"> </P>
</FORM>
</BODY>
</HTML>
```

defaultChecked

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

In the script fragment of Listing 24-7 (not among the CD-ROM files), a function is passed a reference to a form containing the Stooges radio buttons. The goal is to see, in as general a way as possible (supplying the radio group name where needed), if the user changed the default setting. Looping through each of the radio buttons, you look for the one whose `CHECKED` attribute is set in the `<INPUT>` definition. With that index value (`i`) in hand, you then look to see if that entry is still checked. If not (notice the `!` negation operator), you display an alert dialog box about the change.

Listing 24-7: Has a Radio Button Changed?

```
function groupChanged(form) {
    for (var i = 0; i < form.stooges.length; i++) {
        if (form.stooges[i].defaultChecked) {
            if (!form.stooges[i].checked) {
                alert("This radio group has been changed.")
            }
        }
    }
}
```

length

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See the loop construction within the function of Listing 24-7 for one way to apply the `length` property.

value

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 24-6 (earlier in this chapter) demonstrates how a function extracts the `value` property of a radio button to display otherwise hidden information stored with a button. In this case, it lets the alert dialog box show the full name of the selected Stooge.

Event handlers

onClick

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Every time a user clicks one of the radio buttons in Listing 24-8, he or she sets a global variable to `true` or `false`, depending on whether the person is a Shemp lover. This action is independent of the action that is taking place if the user clicks on the View Full Name button. An `onUnload` event handler in the `<BODY>` definition triggers a function that displays a message to Shemp lovers just before the page clears (click the browser's Reload button to leave the current page prior to reloading). Here I use an initialize function triggered by `onLoad` so that the current radio button selection sets the global value upon a reload.

Listing 24-8: An onClick event Handler for Radio Buttons

```
<HTML>
<HEAD>
<TITLE>Radio Button onClick Handler</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var ShempOPhile = false
function initValue() {
    ShempOPhile = document.forms[0].stooges[3].checked
}
 function fullName(form) {
    for (var i = 0; i < form.stooges.length; i++) {
        if (form.stooges[i].checked) {
            break
```

Continued

Listing 24-8 *(continued)*

```
        }
    }
    alert("You chose " + form.stooges[i].value + ".")
}
function setShemp(setting) {
    ShempOPhile = setting
}
function exitMsg() {
    if (ShempOPhile) {
        alert("You like SHEMP?")
    }
}
</SCRIPT>
</HEAD>

<BODY onLoad="initValue()" onUnload="exitMsg()">
<FORM>
<B>Select your favorite Stooge:</B><P>
<INPUT TYPE="radio" NAME="stooges" VALUE="Moe Howard" CHECKED
onClick="setShemp(false)">Moe
<INPUT TYPE="radio" NAME="stooges" VALUE="Larry Fine"
onClick="setShemp(false)">Larry
<INPUT TYPE="radio" NAME="stooges" VALUE="Curly Howard"
onClick="setShemp(false)">Curly
<INPUT TYPE="radio" NAME="stooges" VALUE="Shemp Howard"
onClick="setShemp(true)">Shemp<P>
<INPUT TYPE="button" NAME="Viewer" VALUE="View Full Name..."
onClick="fullName(this.form)">
</FORM>
</BODY>
</HTML>
```

See also Listing 24-5 for further examples of scripting `onClick` event handlers for radio buttons — this time to hide and show related items in a form.

CHAPTER 9

Text-Related Form Objects (Chapter 25)

In This Chapter

Capturing and modifying text field contents

Triggering action and entering text

Giving focus to a text field and selecting its contents

When your page needs input from visitors beyond "yes" or "no" answers, text fields are the interface elements that provide the blank spaces. Whether you specify the one-line INPUT element or the multi-line TEXTAREA element, this is where visitors can not only express themselves, but also enter information in formats that might cause your carefully constructed back-end database to go haywire. More often than not, it is the text box that benefits most from client-side form validation.

Despite the fact that the primary user action in a text box is typing, keyboard events became available to scripters only starting with the version 4 browsers from both Microsoft and Netscape. But they arrived fully formed, with a suite of events for the downstroke, upstroke, and complete press-and-release action of typing a character. From there, the event object takes over to help scripts uncover the character code and whether the user held down any modifier keys while typing the character. You can find examples of this kind of event handling in the examples for Chapters 1 and 13 of this book.

Text boxes are not always as scriptable as you might like them to be. Modern browsers can apply style sheets to adjust font characteristics of the complete text box, but you cannot, say, set some of the words inside a text box to bold. Even something as common (in other programs) as having the text insertion pointer automatically plant itself at the end of existing text is possible so far only in IE4+/Windows via the TEXTAREA's `createTextRange()` method and associated `TextRange` object methods (see `TextRange` object examples in Chapter 5 of this book). The moral of the story is to keep your expectations for the powers of text fields at moderate levels.

Examples Highlights

- ✦ Because the `value` property holds the string value of the text box, it is also the property you use to dump new text into a box. Listings 25-2 and 25-3 read from and write to a text box, transforming the entered contents along the way. You see three different approaches to the task.
- ✦ During client-side validation, you help the visitor by directing the text insertion pointer to the text field that failed a validation. Listing 25-4 shows how to use the `focus()` and `select()` methods along with a workaround for an IE/Windows timing problem that normally gets in the way.
- ✦ Use the `onChange` event handler (not `onBlur`) as a trigger for real-time data validation, as demonstrated in Listing 25-6. You also see the syntax that prevents form submission when validation fails.
- ✦ In IE4+ and NN6, you can adjust the size of a TEXTAREA element after the page has loaded. The example for the `cols` and `rows` properties lets you see the results in The Evaluator.

Text Input Object

Properties

`defaultValue`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Important: Listings 25-1, 25-2, and 25-3 feature a form with only one text INPUT element. The rules of HTML forms say that such a form submits itself if the user presses the Enter key whenever the field has focus. Such a submission to a form whose action is undefined causes the page to reload, thus stopping any scripts that are running at the time. FORM elements for of these example listings contain an `onSubmit` event handler that both blocks the submission and attempts to trigger the text box `onChange` event handler to run the demonstration script. In some browsers, such as IE5/Mac, you may have to press the Tab key or click outside of the text box to trigger the `onChange` event handler after you enter a new value.

Listing 25-1 has a simple form with a single field that has a default value set in its tag. A function (`resetField()`) restores the contents of the page's lone field to the value assigned to it in the `<INPUT>` definition. For a single-field page such as this, defining a `TYPE="reset"` button or calling `form.reset()` works the same way because such buttons reestablish default values of all elements of a form. But if you

want to reset only a subset of fields in a form, follow the example button and function in Listing 25-1.

Listing 25-1: **Resetting a Text Object to Default Value**

```
<HTML>
<HEAD>
<TITLE>Text Object DefaultValue</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function upperMe(field) {
    field.value = field.value.toUpperCase()
}
function resetField(form) {
    form.converter.value = form.converter.defaultValue
}
</SCRIPT>
</HEAD>

<BODY>
<FORM onSubmit="window.focus(); return false">
Enter lowercase letters for conversion to uppercase: <INPUT TYPE="text"
NAME="converter" VALUE="sample" onChange="upperMe(this)">
<INPUT TYPE="button" VALUE="Reset Field"
onClick="resetField(this.form)">
</FORM>
</BODY>
</HTML>
```

form

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The following function fragment receives a reference to a text element as the parameter. The text element reference is needed to decide which branch to follow; then the form is submitted.

```
function setAction(fld) {
    if (fld.value.indexOf("@") != -1) {
        fld.form.action = "mailto:" + fld.value
    } else {
        fld.form.action = "cgi-bin/normal.pl"
    }
    fld.form.submit()
}
```

Notice how this function doesn't have to worry about the form reference, because its job is to work with whatever form encloses the text field that triggers this function.

maxLength

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in *JavaScript Bible*) to experiment with the `maxLength` property. The top text field has no default value, but you can temporarily set it to only a few characters and see how it affects entering new values:

```
document.forms[0].input.maxLength = 3
```

Try typing into the field to see the results of the change. To restore the default value, reload the page.

name

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Consult Listing 25-2 later in this chapter, where I use the text object's name, `convertor`, as part of the reference when assigning a value to the field. To extract the name of a text object, you can use the property reference. Therefore, assuming that your script doesn't know the name of the first object in the first form of a document, the statement is

```
var objectName = document.forms[0].elements[0].name
```

readOnly

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in *JavaScript Bible*) to set the bottom text box to be read-only. Begin by typing anything you want in the bottom text box. Then enter the following statement into the top text box:

```
document.forms[0].inspector.readOnly = true
```

While existing text in the box is selectable (and therefore can be copied into the clipboard), it cannot be modified or removed.

size

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Resize the bottom text box of The Evaluator (Chapter 13 in *JavaScript Bible*) by entering the following statements into the top text box:

```
document.forms[0].inspector.size = 20
document.forms[0].inspector.size = 400
```

Reload the page to return the size back to normal (or set the value to `80`).

value

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

As a demonstration of how to retrieve and assign values to a text object, Listing 25-2 shows how the action in an `onChange` event handler is triggered. Enter any lowercase letters into the field and click out of the field. I pass a reference to the entire form object as a parameter to the event handler. The function extracts the value, converts it to uppercase (using one of the JavaScript string object methods), and assigns it back to the same field in that form.

Listing 25-2: **Getting and Setting a Text Object's Value**

```
<HTML>
<HEAD>
<TITLE>Text Object Value</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function upperMe(form) {
    inputStr = form.converter.value
    form.converter.value = inputStr.toUpperCase()
}
</SCRIPT>
</HEAD>
```

Continued

Listing 25-2 *(continued)*

```
<BODY>
<FORM onSubmit="window.focus(); return false">
Enter lowercase letters for conversion to uppercase: <INPUT TYPE="text"
NAME="converter" VALUE="sample" onChange="upperMe(this.form)">
</FORM>
</BODY>
</HTML>
```

I also show two other ways to accomplish the same task, each one more efficient than the previous example. Both utilize the shortcut object reference to get at the heart of the text object. Listing 25-3 passes the text object — contained in the `this` reference — to the function handler. Because that text object contains a complete reference to it (out of sight, but there just the same), you can access the `value` property of that object and assign a string to that object's `value` property in a simple assignment statement.

Listing 25-3: **Passing a Text Object (as this) to the Function**

```
<HTML>
<HEAD>
<TITLE>Text Object Value</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function upperMe(field) {
    field.value = field.value.toUpperCase()
}
</SCRIPT>
</HEAD>

<BODY>
<FORM onSubmit="window.focus(); return false">
Enter lowercase letters for conversion to uppercase: <INPUT TYPE="text"
NAME="converter" VALUE="sample" onChange="upperMe(this)">
</FORM>
</BODY>
</HTML>
```

Yet another way is to deal with the field values directly in an embedded event handler — instead of calling an external function (which is easier to maintain because all scripts are grouped together in the Head). With the function removed from the document, the event handler attribute of the `<INPUT>` tag changes to do all the work:

```
<INPUT TYPE="text" NAME="converter" VALUE="sample"
onChange="this.value = this.value.toUpperCase()">
```

The right-hand side of the assignment expression extracts the current contents of the field and (with the help of the `toUpperCase()` method of the string object) converts the original string to all uppercase letters. The result of this operation is assigned to the `value` property of the field.

The application of the this keyword in the previous examples may be confusing at first, but these examples represent the range of ways in which you can use such references effectively. Using this by itself as a parameter to an object's event handler refers only to that single object — a text object in Listing 25-3. If you want to pass along a broader scope of objects that contain the current object, use the `this` keyword along with the outer object layer that you want. In Listing 25-2, I sent a reference to the entire form along by specifying `this.form` — meaning the form that contains "this" object, which is being defined in the line of HTML code.

At the other end of the scale, you can use similar-looking syntax to specify a particular property of the `this` object. Thus, in the last example, I zeroed in on just the `value` property of the current object being defined — `this.value`. Although the formats of `this.form` and `this.value` appear the same, the fact that one is a reference to an object and the other just a value can influence the way your functions work. When you pass a reference to an object, the function can read and modify properties of that object (as well as invoke its functions); but when the parameter passed to a function is just a property value, you cannot modify that value without building a complete reference to the object and its value.

Methods

blur()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The following statement invokes the `blur()` method on a text box named `vanishText`:

```
document.forms[0].vanishText.blur()
```

focus()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listing 25-4 for an example of an application of the `focus()` method in concert with the `select()` method.

select()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

A click of the Verify button in Listing 25-4 performs a validation on the contents of the text box, making sure the entry consists of all numbers. All work is controlled by the `checkNumeric()` function, which receives a reference to the field needing inspection as a parameter. Because of the way the delayed call to the `doSelection()` function has to be configured, various parts of what will become a valid reference to the form are extracted from the field's and form's properties. If the validation (performed in the `isNumber()` function) fails, the `setSelection()` method is invoked after an artificial delay of zero milliseconds. As goofy as this sounds, this method is all that IE needs to recover from the display and closure of the alert dialog box. Because the first parameter of the `setTimeout()` method must be a string, the example assembles a string invocation of the `setSelection()` function via string versions of the form and field names. All that the `setSelection()` function does is focus and select the field whose reference is passed as a parameter. This function is now generalizable to work with multiple text boxes in a more complex form.

Listing 25-4: **Selecting a Field**

```
<HTML>
<HEAD>
<TITLE>Text Object Select/Focus</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// general purpose function to see if a suspected numeric input is a number
function isNumber(inputStr) {
    for (var i = 0; i < inputStr.length; i++) {
        var oneChar = inputStr.charAt(i)
        if (oneChar < "0" || oneChar > "9") {
            alert("Please make sure entries are integers only.")
            return false
        }
    }
    return true
}
function checkNumeric(fld) {
    var inputStr = fld.value
    var fldName = fld.name
    var formName = fld.form.name
    if (isNumber(inputStr)) {
        // statements if true
    } else {
```

```
        setTimeout("doSelection(document." + formName + ". " + fldName + ")", 0)
    }
}

function doSelection(fld) {
    fld.focus()
    fld.select()
}
</SCRIPT>
</HEAD>

<BODY>
<FORM NAME="entryForm" onSubmit="return false">
Enter any positive integer: <INPUT TYPE="text" NAME="numeric"><P>
<INPUT TYPE="button" VALUE="Verify" onClick="checkNumeric(this.form.numeric)">
</FORM>
</BODY>
</HTML>
```

Event handlers

onBlur
onFocus
onSelect

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

To demonstrate one of these event handlers, Listing 25-5 shows how you may use the window's statusbar as a prompt message area after a user activates any field of a form. When the user tabs to or clicks on a field, the prompt message associated with that field appears in the statusbar. In Figure 9-1, the user has tabbed to the second text box, which caused the statusbar message to display a prompt for the field.

Listing 25-5: **The onFocus event Handler**

```
<HTML>
<HEAD>
<TITLE>Elements Array</TITLE>
<SCRIPT LANGUAGE="JavaScript">
```

Continued

Listing 25-5 *(continued)*

```
function prompt(msg) {
    window.status = "Please enter your " + msg + "."
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
Enter your first name:<INPUT TYPE="text" NAME="firstName"
onFocus="prompt('first name')"><P>
Enter your last name:<INPUT TYPE="text" NAME="lastName"
onFocus="prompt('last name')"><P>
Enter your address:<INPUT TYPE="text" NAME="address"
onFocus="prompt('address')"><P>
Enter your city:<INPUT TYPE="text" NAME="city" onFocus="prompt('city')"><P>
</FORM>
</BODY>
</HTML>
```

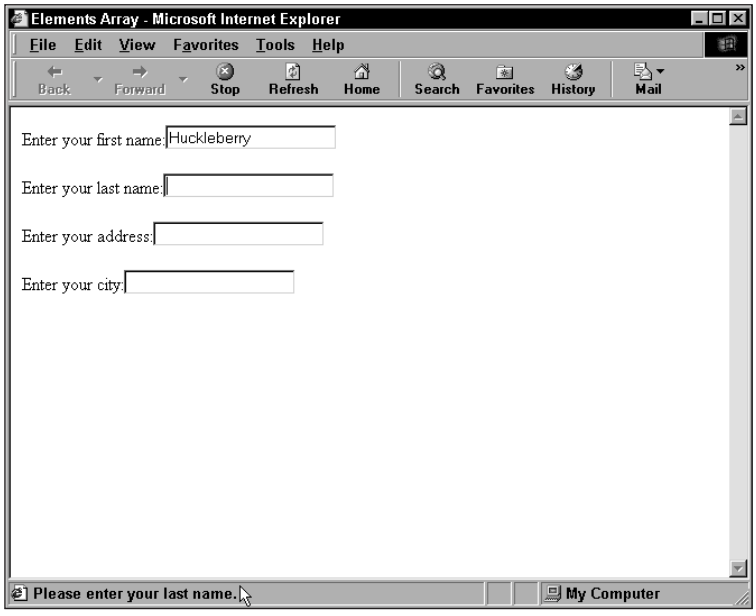


Figure 9-1: An onFocus event handler triggers a statusbar display.

onChange

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Whenever a user makes a change to the text in a field in Listing 25-6 and then either tabs or clicks out of the field, the change event is sent to that field, triggering the `onChange` event handler.

Because the form in Listing 25-6 has only one field, the example demonstrates a technique you can use that prevents a form from being "submitted" if the user accidentally presses the Enter key. The technique is as simple as defeating the submission via the `onSubmit` event handler of the form. At the same time, the `onSubmit` event handler invokes the `checkIt()` function, so that pressing the Enter key (as well as pressing Tab or clicking outside the field) triggers the function.

Listing 25-6: **Data Validation via an onChange event Handler**

```
<HTML>
<HEAD>
<TITLE>Text Object Select/Focus</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// general purpose function to see if a suspected numeric input is a number
function isNumber(inputStr) {
    for (var i = 0; i < inputStr.length; i++) {
        var oneChar = inputStr.substring(i, i + 1)
        if (oneChar < "0" || oneChar > "9") {
            alert("Please make sure entries are numbers only.")
            return false
        }
    }
    return true
}
function checkIt(form) {
    inputStr = form.numeric.value
    if (isNumber(inputStr)) {
        // statements if true
    } else {
        form.numeric.focus()
        form.numeric.select()
    }
}
</SCRIPT>
</HEAD>

<BODY onSubmit="checkIt(this); return false">
```

Continued

Listing 25-6 *(continued)*

```
<FORM>
Enter any positive integer: <INPUT TYPE="text" NAME="numeric"
onChange="checkIt(this.form)"><P>
</FORM>
</BODY>
</HTML>
```

TEXTAREA Element Object

Properties

`cols`
`rows`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator to play with the `cols` and `rows` property settings for the Results textarea on that page. Shrink the width of the textarea by entering the following statement into the top text box:

```
document.forms[0].output.cols = 30
```

And make the textarea one row deeper:

```
document.forms[0].output.rows++
```

Methods

`createTextRange()`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See the example for the `TextRange.move()` method in Chapter 5 of this book to see how to control the text insertion pointer inside a TEXTAREA element.

✦ ✦ ✦

CHAPTER 10

Select, Option, and Optgroup Objects (Chapter 26)

In This Chapter

Triggering action based on a user's selection in a pop-up or select list

Reading hidden and visible values of OPTION element items

Scripting SELECT objects that allow multiple selections

The SELECT element is the best space-saving device in the HTML form repertoire. Whether you choose the pop-up menu or scrolling list display style, your page can provide visitors with a visually compact list of literally hundreds of items from which to choose. From a scripter's point of view, however, it is a complex item to manage, especially in older browsers.

In truth, the SELECT element is an outer wrapper for the OPTION element items nested within. Each OPTION element contains the text that the user sees in the list, as well as a hidden value that may be more meaningful to a server database or client script. The difficulty with browsers prior to IE4 and NN6 is that reading the hidden value of the currently chosen item in the list requires an extensive reference to not only the SELECT element, but to the item in the array of OPTION element objects. To reach that specific item, the script uses a reference to the SELECT object's `selectedIndex` property as the `options` array index. Newer browsers simplify the matter by providing a single `value` property for the SELECT object that returns the value of the currently selected item (or of the first item when multiple choices are allowed).

Many browser versions provide script facilities for modifying the content of a SELECT list. But the effect is not perfect in browsers that don't also reflow the page to reflect the potentially resized width of the list.

A user interface debate rages about whether a SELECT list, whose purpose is obviously intended to direct site navigation, should navigate immediately upon making a choice or if the user should also click on an explicit "Go" button next to the list. The former is faster for the impatient visitor, but the latter doesn't shoot off to an undesired page when the user makes a wrong selection. Good luck with that decision.

Examples Highlights

- ✦ To harvest the values of all selected items in a multiple list, your script needs to cycle through the SELECT element's `options` array and inspect the `selected` property of each, as shown in Listing 26-4.
- ✦ Scripts can also retrieve the text of the selected item, instead of the hidden value. Compare two similar applications that work with the `text` (Listing 26-5) and `value` (Listing 26-6) properties.
- ✦ Listings 26-5 and 26-6 show the backward-compatible, long reference to retrieve a chosen option's details. The modern alternative accompanies the example for the `SELECT.value` property.
- ✦ See Listing 26-8 for another example of triggering a script via the `onChange` event handler of a SELECT object.
- ✦ Implementations of the OPTGROUP element object may need improvement before Listing 26-9 behaves as it should to modify hierarchical labels within a SELECT list.

SELECT Element Object

Properties

`length`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listing 26-1 in Chapter 26 of the *JavaScript Bible* for an illustration of the way you use the `length` property to help determine how often to cycle through the repeat loop in search of selected items. Because the loop counter, `i`, must start at `0`, the counting continues until the loop counter is one less than the actual length value (which starts its count with `1`).

`multiple`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statement toggles between single and multiple selections on a SELECT element object whose `SIZE` attribute is set to a value greater than `1`:

```
document.forms[0].mySelect.multiple = !document.forms[0].mySelect.multiple
```

options[*index*]

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

See Listings 26-1 through 26-3 in Chapter 26 of the *JavaScript Bible* for examples of how the `options` array references information about the options inside a SELECT element.

options[*index*].defaultSelected

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The following statement preserves a Boolean value if the first option of the SELECT list is the default selected item:

```
var zeroIsDefault = document.forms[0].listName.options[0].defaultSelected
```

options[*index*].index

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

The following statement assigns the index integer of the first option of a SELECT element named `listName` to a variable named `itemIndex`.

```
var itemIndex = document.forms[0].listName.options[0].index
```

options[*index*].selected

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

To accumulate a list of all items selected by the user, the `seeList()` function in Listing 26-4 systematically examines the `options[index].selected` property of each item in the list. The text of each item whose `selected` property is `true` is appended to the list. I add the `"\n "` inline carriage returns and spaces to make the list in the alert dialog box look nice and indented. If you assign other values to the `VALUE` attributes of each option, the script can extract the `options[index].value` property to collect those values instead.

Listing 26-4: **Cycling through a Multiple-Selection List**

```
<HTML>
<HEAD>
<TITLE>Accessories List</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function seeList(form) {
    var result = ""
    for (var i = 0; i < form.accList.length; i++) {
        if (form.accList.options[i].selected) {
            result += "\n  " + form.accList.options[i].text
        }
}
    alert("You have selected:" + result)
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<P>Control/Command-click on all accessories you use:
<SELECT NAME="accList" SIZE=9 MULTIPLE>
    <OPTION SELECTED>Color Monitor
    <OPTION>Modem
    <OPTION>Scanner
    <OPTION>Laser Printer
    <OPTION>Tape Backup
    <OPTION>MO Drive
    <OPTION>Video Camera
</SELECT> </P>
<P><INPUT TYPE="button" VALUE="View Summary..."
onClick="seeList(this.form)"></P>
</FORM>
</BODY>
</HTML>
```

options[*index*].text

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

To demonstrate the `text` property of an option, Listing 26-5 applies the text from a selected option to the `document.bgColor` property of a document in the current window. The color names are part of the collection built into all scriptable browsers; fortunately, the values are case-insensitive so that you can capitalize the color names displayed and assign them to the property.

Listing 26-5: Using the options[index].text Property

```
<HTML>
<HEAD>
<TITLE>Color Changer 1</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function seeColor(form) {
    var newColor = (form.colorsList.options[form.colorsList.selectedIndex].text)
    document.bgColor = newColor
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<P>Choose a background color:
<SELECT NAME="colorsList">
    <OPTION SELECTED>Gray
    <OPTION>Lime
    <OPTION>Ivory
    <OPTION>Red
</SELECT></P>
<P><INPUT TYPE="button" VALUE="Change It" onClick="seeColor(this.form)"></P>
</FORM>
</BODY>
</HTML>
```

options[*index*].value

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 26-6 requires the option text that the user sees to be in familiar, multiple-word form. But to set the color using the browser's built-in color palette, you must use the one-word form. Those one-word values are stored in the `VALUE` attributes of each `<OPTION>` definition. The function then reads the `value` property, assigning it to the `bgColor` of the current document. If you prefer to use the hexadecimal triplet form of color specifications, those values are assigned to the `VALUE` attributes (`<OPTION VALUE="#e9967a">Dark Salmon`).

Listing 26-6: **Using the options[index].value Property**

```
<HTML>
<HEAD>
<TITLE>Color Changer 2</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function seeColor(form) {
    var newColor =
(form.colorsList.options[form.colorsList.selectedIndex].value)
    document.bgColor = newColor
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<P>Choose a background color:
<SELECT NAME="colorsList">
    <OPTION SELECTED VALUE="cornflowerblue">Cornflower Blue
    <OPTION VALUE="darksalmon">Dark Salmon
    <OPTION VALUE="lightgoldenrodyellow">Light Goldenrod Yellow
    <OPTION VALUE="seagreen">Sea Green
</SELECT></P>
<P><INPUT TYPE="button" VALUE="Change It" onClick="seeColor(this.form)"></P>
</FORM>
</BODY>
</HTML>
```

selectedIndex

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

In the `inspect()` function of Listing 26-7, notice that the value inside the `options` property index brackets is a reference to the object's `selectedIndex` property. Because this property always returns an integer value, it fulfills the needs of the index value for the `options` property. Therefore, if you select Green in the pop-up menu, `form.colorsList.selectedIndex` returns a value of `1`; that reduces the rest of the reference to `form.colorsList.options[1].text`, which equals "Green."

Listing 26-7: **Using the selectedIndex Property**

```
<HTML>
<HEAD>
<TITLE>Select Inspector</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function inspect(form) {
    alert(form.colorsList.options[form.colorsList.selectedIndex].text)
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<P><SELECT NAME="colorsList">
    <OPTION SELECTED>Red
    <OPTION VALUE="Plants"><I>Green</I>
    <OPTION>Blue
</SELECT></P>
<P><INPUT TYPE="button" VALUE="Show Selection" onClick="inspect(this.form)"></P>
</FORM>
</BODY>
</HTML>
```

size

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statement sets the number of visible items to 5:

```
document.forms[0].mySelect.size = 5
```

value

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The function in Listing 26-6 that accesses the chosen value the long way can be simplified for newer browsers only with the following construction:

```
function seeColor(form) {
    document.bgColor = form.colorsList.value
}
```

Methods

item(*index*)
namedItem("*optionID*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

The following statement assigns an OPTION element reference to a variable:

```
var oneOption = document.forms[0].mySelect.namedItem("option3_2")
```

Event handlers

onChange

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 26-8 is a version of Listing 26-6 that invokes all action as the result of a user making a selection from the pop-up menu. The `onChange` event handler in the `<SELECT>` tag replaces the action button. For this application — when you desire a direct response to user input — an appropriate method is to have the action triggered from the pop-up menu rather than by a separate action button.

Notice two other important changes. First, the SELECT element now contains a blank first option. When a user visits the page, nothing is selected yet, so you should present a blank option to encourage the user to make a selection. The function also makes sure that the user selects one of the color-valued items before it attempts to change the background color.

Second, the BODY element contains an `onUnload` event handler that resets the form. The purpose behind this is that if the user navigates to another page and uses the Back button to return to the page, the script-adjusted background color does not persist. I recommend you return the SELECT element to its original setting. Unfortunately, the reset does not stick to the form in IE4 and IE5 for Windows (although this problem appears to be repaired in IE5.5). Another way to approach this issue is to use the `onLoad` event handler to invoke `seeColor()`, passing as a parameter a reference to the SELECT element. Thus, if the SELECT element choice persists, the background color is adjusted accordingly after the page loads.

Listing 26-8: **Triggering a Color Change from a Pop-Up Menu**

```
<HTML>
<HEAD>
<TITLE>Color Changer 2</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function seeColor(list) {
    var newColor = (list.options[list.selectedIndex].value)
    if (newColor) {
        document.bgColor = newColor
    }
}
</SCRIPT>
</HEAD>

<BODY onUnload="document.forms[0].reset()">
<FORM>
```

Continued

Listing 26-8 *(continued)*

```
<P>Choose a background color:
<SELECT NAME="colorsList" onChange="seeColor(this)">
    <OPTION SELECTED VALUE="">
    <OPTION VALUE="cornflowerblue">Cornflower Blue
    <OPTION VALUE="darksalmon">Dark Salmon
    <OPTION VALUE="lightgoldenrodyellow">Light Goldenrod Yellow
    <OPTION VALUE="seagreen">Sea Green
</SELECT></P>
</FORM>
</BODY>
</HTML>
```

OPTION Element Object

Properties

label

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	

Example

The following statement modifies the text that appears as the selected text in a pop-up list:

```
document.forms[0].mySelect.options[3].label = "Widget 9000"
```

If this option is the currently selected one, the text on the pop-up list at rest changes to the new label.

OPTGROUP Element Object

Properties

label

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				✓	

Example

I present Listing 26-9 in the hope that Microsoft and Netscape will eventually eradicate the bugs that afflict their current implementations of the `label` property. When the feature works as intended, Listing 26-9 demonstrates how a script can alter the text of option group labels. This page is an enhanced version of the background color setters used in other examples of this chapter. Be aware that IE5/Mac does not alter the last OPTGROUP element's label, and NN6 achieves only a partial change to the text displayed in the SELECT element.

Listing 26-9: **Modifying OPTGROUP Element Labels**

```
<HTML>
<HEAD>
<TITLE>Color Changer 3</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var regularLabels = ["Reds","Greens","Blues"]
var naturalLabels = ["Apples","Leaves","Sea"]
function setRegularLabels(list) {
    var optGrps = list.getElementsByTagName("OPTGROUP")
    for (var i = 0; i < optGrps.length; i++) {
        optGrps[i].label = regularLabels[i]
    }
}
function setNaturalLabels(list) {
    var optGrps = list.getElementsByTagName("OPTGROUP")
    for (var i = 0; i < optGrps.length; i++) {
       optGrps[i].label = naturalLabels[i]
    }
}
function seeColor(list) {
    var newColor = (list.options[list.selectedIndex].value)
    if (newColor) {
        document.bgColor = newColor
    }
}
</SCRIPT>
</HEAD>

<BODY onUnload="document.forms[0].reset()">
<FORM>
<P>Choose a background color:
<SELECT name="colorsList" onChange="seeColor(this)">
    <OPTGROUP ID="optGrp1" label="Reds">
        <OPTION value="#ff9999">Light Red
        <OPTION value="#ff3366">Medium Red
        <OPTION value="#ff0000">Bright Red
        <OPTION value="#660000">Dark Red
    </OPTGROUP>
    <OPTGROUP ID="optGrp2" label="Greens">
        <OPTION value="#ccff66">Light Green
```

Continued

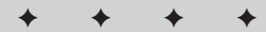
Listing 26-9 *(continued)*

```
        <OPTION value="#99ff33">Medium Green
        <OPTION value="#00ff00">Bright Green
        <OPTION value="#006600">Dark Green
    </OPTGROUP>
    <OPTGROUP ID="optGrp3" label="Blues">
        <OPTION value="#ccffff">Light Blue
        <OPTION value="#66ccff">Medium Blue
        <OPTION value="#0000ff">Bright Blue
        <OPTION value="#000066">Dark Blue
    </OPTGROUP>
</SELECT></P>
<P>
<INPUT TYPE="radio" NAME="labels" CHECKED
onClick="setRegularLabels(this.form.colorsList)">Regular Label Names
<INPUT TYPE="radio" NAME="labels"
onClick="setNaturalLabels(this.form.colorsList)">Label Names from Nature</P>
</FORM>
</BODY>
</HTML>
```

✦ ✦ ✦

CHAPTER 11

Table and List Objects (Chapter 27)



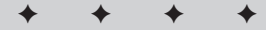
In This Chapter

Modifying table cell content

Adding and deleting table rows

Modifying table dimensions, colors, and borders

Changing numbering sequences and bullet symbols for LI element objects



Dynamic object models that take advantage of automatic page reflow create huge opportunities for creative Web designers. Nowhere is that more apparent than in the TABLE element object and all the other objects that nest within (TR, TH, TD, and so on). Not only is it possible to swap the content of a table cell at any time, but the object models provide powerful methods for completely remolding the composition of a table on the fly.

HTML tables are at once elegant because they provide a lot of pleasing organization to a page with little code, and also complex due to the large number of related elements and substantial list of attributes for each element. Those attributes become object properties in the modern object model, so it means that scripters have much to choose from (and be confused by) when bringing tables to life.

Using the special-purpose methods that insert rows and cells also takes some initial adjustment for many scripters. For example, inserting a row has almost no visual effect on an existing table until you not only insert cells into the row, but also plant content in the cells. Code examples for these operations are part of the general discussion of the TABLE object in the *JavaScript Bible*.

Designers whose browser targets are IE4+/Windows can also take advantage of Microsoft's data binding technology. Data from external sources can fill tables with only the slightest bit of HTML markup. Chapter 15 contains examples of this in its discussion of the `dataFld` and related properties.

This chapter also includes objects for ordered and unordered lists (and list items nested within). In concert with style sheets that can include or exclude elements from page rendering, these objects provide additional layout opportunities for clever designers.

Examples Highlights

- Scripts can adjust the value of a TABLE object's `width` property, including switching between a fixed pixel size and a percentage of the table container's width.
- Compare the examples for the IE5/Windows `TABLE.cells` property and the `TR.cells` property for IE4+ and NN6.
- Follow the example for the `TD.colSpan` property to observe how a table responds to such changes in real time.
- Examples for list-related elements show how to set the list types for script-generated lists.

TABLE Element Object

Properties

align

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see the `align` property at work. The default value (`left`) is in force when the page loads. But you can shift the table to right-align with the body by entering the following statement into the top text box for IE5+ and NN6+:

```
document.getElementById("myTable").align = "right"
```

background

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Treat the `background` property of a table like you do the `src` property of an IMG element object. If you precache an image, you can assign the `src` property of the precached `image` object to the `background` property of the table for quick image changing. Such an assignment statement looks like the following:

```
document.all.myTable.background = imgArray["myTableAlternate"].src
```

bgColor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to assign a color to the table. After looking at the table to see its initial state, enter the following IE5+/NN6+ statement into the top text box:

```
document.getElementById("myTable").bgColor = "lightgreen"
```

When you look at the table again, you see that only some of the cells turned to green. This is because colors also are assigned to table elements nested inside the outermost table element, and the color specification closest to the actual element wins the contest.

border

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

To remove all traces of an outside border of a table (and, in some combinations of attributes of other table elements, borders between cells), use the following statement (in IE5+/NN6+ syntax):

```
document.getElementById("myTable").border = 0
```

borderColor
borderColorDark
borderColorLight

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Assuming that you have set the initial light and dark color attributes of a table, the following function swaps the light and dark colors to shift the light source to the opposite corner:

```
function swapColors(tableRef) {
    var oldLight = tableRef.borderColorLight
    tableRef.borderColorLight = tableRef.borderColorDark
    tableRef.borderColorDark = oldLight
}
```

While you can easily invoke this function over and over by ending it with a `setTimeout()` method that calls this function after a fraction of a second, the results are very distracting to the person trying to read your page. Please don't do it.

caption

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following example, for use with The Evaluator (Chapter 13 in the *JavaScript Bible*) in NN6+, demonstrates the sequence of assigning a new CAPTION element object to a table. While the table in The Evaluator already has a CAPTION element, the following statements replace it with an entirely new one. Enter each of the following statements into the top text box, starting with the one that saves a long reference into a variable for multiple uses at the end:

```
t = document.getElementById("myTable")
a = document.createElement("CAPTION")
b = document.createTextNode("A Brand New Caption")
a.appendChild(b)
t.replaceChild(a, t.caption)
```

A view of the table shows that the new caption has replaced the old one because a table can have only one CAPTION element.

cellPadding
cellSpacing

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to adjust the `cellPadding` and `cellSpacing` properties of the demonstrator table. First, adjust the padding (IE5+/NN6 syntax):

```
document.getElementById("myTable").cellPadding = 50
```

Now, adjust the cell spacing:

```
document.getElementById("myTable").cellSpacing = 15
```

Notice how `cellSpacing` affected the thickness of inter-cell borders.

cells

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator with IE5+ for Windows to have JavaScript calculate the number of columns in the demonstrator table with the help of the `cells` and `rows` properties. Enter the following statement into the top text box:

```
document.all.myTable.cells.length/document.all.myTable.rows.length
```

The result is the number of columns in the table.

dataPageSize

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

If you want to change the number of visible rows of linked data in the table to 15, use the following statement:

```
document.all.myTable.dataPageSize = 15
```

frame

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listing 27-4 presents a page that cycles through all possible settings for the `frame` property. The `frame` property value is displayed in the table's caption. (Early versions of NN6 might fail to refresh part of the page after adjusting the `frame` property.)

Listing 27-4: **Cycling Through Table frame Property Values**

```
<HTML>
<HEAD>
<TITLE>TABLE.frame Property</TITLE>

<SCRIPT LANGUAGE="JavaScript">
var timeoutID
var frameValues = ["box", "above", "rhs", "below", "lhs", "hsides", "vsides",
                   "border", "void"]
function rotateBorder(i) {
    document.getElementById("myTABLE").frame = frameValues[i]
    document.getElementById("myCAPTION").innerHTML = frameValues[i]
    i = (++i == frameValues.length) ? 0 : i
    timeoutID = setTimeout("rotateBorder(" + i + ")", 2000)
}
function stopRotate() {
    clearTimeout(timeoutID)
    document.getElementById("myTABLE").frame = "box"
    document.getElementById("myCAPTION").innerHTML = "box"
}
</SCRIPT>
</HEAD>

<BODY>
<H1>TABLE.frame Property</H1>
<HR>
<FORM NAME="controls">
<FIELDSET>
<LEGEND>Cycle Table Edge Visibility</LEGEND>
<TABLE WIDTH="100%" CELLSPACING=20><TR>
<TD><INPUT TYPE="button" VALUE="Cycle" onClick="rotateBorder(0)"></TD>
<TD><INPUT TYPE="button" VALUE="Stop" onClick="stopRotate()"></TD>
</TR>
</TABLE>
</FIELDSET>
</TABLE>
</FIELDSET>
</FORM>
<HR>
<TABLE ID="myTABLE" CELLPADDING=5 BORDER=3 ALIGN="center">
<CAPTION ID="myCAPTION">Default</CAPTION>
<THEAD ID="myTHEAD">
<TR>
    <TH>River<TH>Outflow<TH>Miles<TH>Kilometers
</TR>
</THEAD>
<TBODY>
<TR>
    <TD>Nile<TD>Mediterranean<TD>4160<TD>6700
</TR>
<TR>
    <TD>Congo<TD>Atlantic Ocean<TD>2900<TD>4670
```

```
</TR>
<TR>
    <TD>Niger<TD>Atlantic Ocean<TD>2600<TD>4180
</TR>
<TR>
    <TD>Zambezi<TD>Indian Ocean<TD>1700<TD>2740
</TR>
</TABLE>
</BODY>
</HTML>
```

height
width

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to adjust the width of the demonstrator table. Begin by increasing the width to the full width of the page:

```
document.getElementById("myTable").width = "100%"
```

To restore the table to its minimum width, assign a very small value to the property:

```
document.getElementById("myTable").width = 50
```

If you have IE4+, you can perform similar experiments with the `height` property of the table.

rows

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator to examine the number of rows in the demonstrator table. Enter the following statement into the top text box:

```
document.getElementById("myTable").rows.length
```

In contrast, notice how the `rows` property sees only the rows within the demonstrator table's TBODY element:

```
document.getElementById("myTbody").rows.length
```

rules

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Listing 27-5 presents a page that cycles through all possible settings for the `rules` property. The `rules` property value is displayed in the table's caption. When you run this script, notice the nice border display for this table's combination of COLGROUP and table row segment elements. Figure 11-1 shows the IE/Windows rendition for the `groups` type of table rules. Early versions of NN6 may not render the altered table correctly, and scripted changes won't appear on the page.

Listing 27-5: Cycling Through Table rules Property Values

```
<HTML>
<HEAD>
<TITLE>TABLE.rules Property</TITLE>

<SCRIPT LANGUAGE="JavaScript">
var timeoutID
var rulesValues = ["all", "cols", "groups", "none", "rows"]
function rotateBorder(i) {
    document.getElementById("myTABLE").rules = rulesValues[i]
    document.getElementById("myCAPTION").innerHTML = rulesValues[i]
    i = (++i == rulesValues.length) ? 0 : i
    timeoutID = setTimeout("rotateBorder(" + i + ")", 2000)
}
function stopRotate() {
    clearTimeout(timeoutID)
    document.getElementById("myTABLE").rules = "all"
    document.getElementById("myCAPTION").innerHTML = "all"
}
</SCRIPT>
</HEAD>

<BODY>
<H1>TABLE.rules Property</H1>
<HR>
<FORM NAME="controls">
<FIELDSET>
<LEGEND>Cycle Table Rule Visibility</LEGEND>
<TABLE WIDTH="100%" CELLSPACING=20><TR>
<TD><INPUT TYPE="button" VALUE="Cycle" onClick="rotateBorder(0)"></TD>
<TD><INPUT TYPE="button" VALUE="Stop" onClick="stopRotate()"></TD>
</TR>
</TABLE>
</FIELDSET>
</TABLE>
</FIELDSET>
</FORM>
```

```
<HR>
<TABLE ID="myTABLE" CELLPADDING=5 BORDER=3 ALIGN="center">
<CAPTION ID="myCAPTION">Default</CAPTION>
<COLGROUP SPAN=1>
<COLGROUP SPAN=3>
<THEAD ID="myTHEAD">
<TR>
    <TH>River<TH>Outflow<TH>Miles<TH>Kilometers
</TR>
</THEAD>
<TBODY>
<TR>
    <TD>Nile<TD>Mediterranean<TD>4160<TD>6700
</TR>
<TR>
    <TD>Congo<TD>Atlantic Ocean<TD>2900<TD>4670
</TR>
<TR>
    <TD>Niger<TD>Atlantic Ocean<TD>2600<TD>4180
</TR>
<TR>
    <TD>Zambezi<TD>Indian Ocean<TD>1700<TD>2740
</TR>
</TABLE>
</BODY>
</HTML>
```

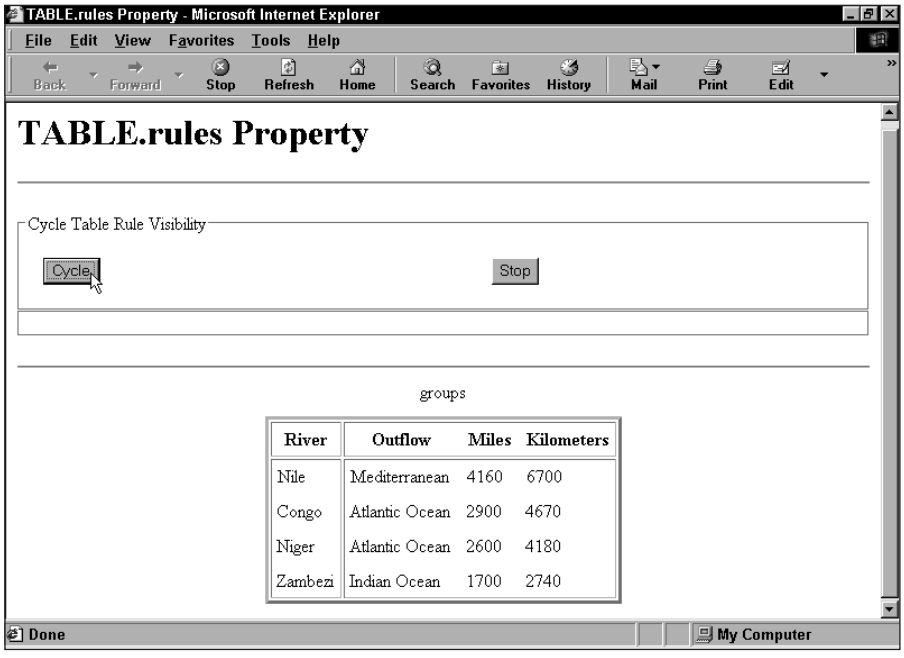


Figure 11-1: The TABLE.rules property set to "groups"

tBodies

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to access the `tBodies` array and reveal the number of rows in the one TBODY segment of the demonstrator table. Enter the following statement into the top text box:

```
document.getElementById("myTable").tBodies[0].rows.length
```

Methods

moveRow(*sourceRowIndex, destinationRowIndex*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

If you want to shift the bottom row of a table to the top, you can use the shortcut reference to the last item's index value (`-1`) for the first parameter:

```
var movedRow = document.all.someTable.moveRow(-1, 0)
```

TBODY, TFOOT, and THEAD Element Objects

Properties

vAlign

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to modify the vertical alignment of the content of the TBODY element in the demonstrator table. Enter the following statement in the top text box to shift the content to the bottom of the cells:

```
document.getElementById("myTBody").vAlign = "bottom"
```

Notice that the cells of the THEAD element are untouched by the action imposed on the TBODY element.

COL and COLGROUP Element Objects

Properties

span

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statement assigns a `span` of `3` to a newly created COLGROUP element stored in the variable `colGroupA`:

```
colGroupA.span = 3
```

TR Element Object

Properties

cells

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to retrieve the number of TD elements in the second row of the demonstrator table. Enter the following statement into the top text box (W3C DOM syntax shown here):

```
document.getElementById("myTable").rows[1].cells.length
```

height

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) in IE4+ to expand the height of the second row of the demonstrator table. Enter the following statement into the top text box:

```
document.all.myTable.rows[1].height = 300
```

If you attempt to set the value very low, the rendered height goes no smaller than the default height.

rowIndex
sectionRowIndex

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to explore the `rowIndex` and `sectionRowIndex` property values for the second physical row in the demonstrator table. Enter each of the following statements into the top text box (W3C DOM syntax shown here):

```
document.getElementById("myTable").rows[1].rowIndex
document.getElementById("myTable").rows[1].sectionRowIndex
```

The result of the first statement is `1` because the second row is the second row of the entire table. But the `sectionRowIndex` property returns `0` because this row is the first row of the TBODY element in this particular table.

TD and TH Element Objects

Properties

cellIndex

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

You can rewrite the cell addition portion of Listing 27-2 (in Chapter 27 in the *JavaScript Bible*) to utilize the `cellIndex` property. The process entails modifying the `insertTableRow()` function so that it uses a `do...while` construction to keep adding cells to match the number of data slots. The function looks like the following (changes shown in boldface):

```
function insertTableRow(form, where) {
    var now = new Date()
    var nowData = [now.getHours(), now.getMinutes(), now.getSeconds(),
        now.getMilliseconds()]
    clearBGColors()
    var newCell
    var newRow = theTableBody.insertRow(where)
    var i = 0
    do {
        newCell = newRow.insertCell(i)
        newCell.innerHTML = nowData[i++]
        newCell.style.backgroundColor = "salmon"
    } while (newCell.cellIndex < nowData.length)
    updateRowCounters(form)
}
```

This version is merely for demonstration purposes and is not as efficient as the sequence shown in Listing 27-2. But the `cellIndex` property version can give you some implementation ideas for the property. It also shows how dynamic the property is, even for brand new cells.

colSpan
rowSpan

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to witness how modifying either of these properties in an existing table can destroy the table. Enter the following statement into the top text box:

```
document.getElementById("myTable").rows[1].cells[0].colSpan = 3
```

Now that the first cell of the second row occupies the space of three columns, the browser has no choice but to shift the two other defined cells for that row out beyond the original boundary of the table. Experiment with the `rowSpan` property the same way. To restore the original settings, assign `1` to each property.

height
width

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see the results of setting the `height` and `width` properties of an existing table cell. Enter each of the following statements into the top text box and study the results in the demonstrator table (W3C DOM syntax used here):

```
document.getElementById("myTable").rows[1].cell[1].height = 100
document.getElementById("myTable").rows[2].cell[0].width = 300
```

You can restore both cells to their original sizes by assigning very small values, such as `1` or `0`, to the properties. The browser prevents the cells from rendering any smaller than is necessary to show the content.

noWrap

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statement creates a new cell in a row and sets its `noWrap` property to prevent text from word-wrapping inside the cell:

```
newCell = newRow.insertCell(-1)
newCell.noWrap = true
```

You need to set this property only if the cell must behave differently than the default, word-wrapping style.

OL Element Object

Properties

start

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statements generate a new OL element and assign a value to the `start` property:

```
var newOL = document.createElement("OL")
newOL.start = 5
```

type

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statements generate a new OL element and assign a value to the `type` property so that the sequence letters are uppercase Roman numerals:

```
var newOL = document.createElement("OL")
newOL.type = "I"
```

UL Element Object

Properties

type

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statements generate a new UL element and assign a value to the `type` property so that the bullet characters are empty circles:

```
var newUL = document.createElement("UL")
newUL.type = "circle"
```

LI Element Object

Properties

type

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

See the examples for the `OL.type` and `UL.type` properties earlier in this chapter.

value

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

The following statements generate a new LI element and assign a value to the `start` property:

```
var newLI = document.createElement("LI")
newLI.start = 5
```

✦ ✦ ✦

CHAPTER 12

Navigator and Other Environment Objects (Chapter 28)

In This Chapter

Determining the user's browser, operating system, and video monitor settings

Modifying NN4+ browser preferences

Retrieving IE4+ user profile information

Objects covered in this chapter are somewhat distant from the document and its content, but they are no less important to scripters. Any script branching that relies on knowing details about the browser version or other aspects of the environment running the browser calls upon the `navigator` object. Properties of the `navigator` object (also named the `clientInformation` object in IE4+), reveal browser brand and version information, as well as operating system and, in some cases, encryption powers of the browser. Using signed scripts with NN4+, you can even script modification to browser preferences.

Avoid using `navigator` object properties for browser version branching when more sophisticated techniques — notably object detection as described in Chapter 14 of the *JavaScript Bible* — are less dependent upon future quirks in object model developments. But version detection is perfect when you know that a special workaround is needed for some glitch in a specific version or class of browser. For example, NN4/Windows can exhibit some strange behavior when attempting to print a page whose content relies on script execution. Provided you have a code workaround for the problem, you can divert script execution for just that version of NN in just the Windows version.

Examples in this chapter also touch upon the `screen` object and the IE/Windows `userProfile` object. The `screen` object is useful in determining the size of a new window, but there is little need to script the `userProfile` object.

Examples Highlights

- Listing 28-1 provides numerous functions that examine `navigator` object properties. The functions examples are provided more as demonstrations of specific values your scripts may need to look for, rather than as some super "browser sniffer." Determining specific IE versions is a bit tricky, so observe how to go about it by way of the `navigator.appVersion` property.
- NN4+ provides access to browser preferences via the `navigator.preference()` method, as shown in Listing 28-2. To implement this feature in a production page, you'll need to use signed scripts.
- Experiment with the `screen.availLeft` and `screen.availTop` properties in NN4+, especially in the Windows environment to see how the taskbar affects these property values.
- For IE4+/Windows, follow the sequence of examples for the `userProfile` object's methods to see how scripts can read user profile fields.

clientInformation Object (IE4+) and navigator Object (All)

Properties

```
appCodeName
appName
appVersion
userAgent
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 28-1 provides a number of reusable functions that your scripts can employ to determine a variety of information about the currently running browser. This is not intended in any way to be an all-inclusive browser-sniffing routine; instead, I offer samples of how to extract information from the key navigator properties to determine various browser conditions.

All functions in Listing 28-1 return a Boolean value inline with the pseudo-question presented in the function's name. For example, the `isWindows()` function returns `true` if the browser is any type of Windows browser; otherwise, it returns `false`. (In Internet Explorer 3, the values are `0` for `false` and `-1` for `true`, but those

values are perfectly usable in `if` conditional phrases). If this kind of browser detection occurs frequently in your pages, consider moving these functions into an external `.js` source library for inclusion in your pages (see Chapter 13 of the *JavaScript Bible* for tips on creating `.js` libraries). When you load this page, it presents fields that display the results of each function depending on the type of browser and client operating system you use.

Listing 28-1: **Functions to Examine Browsers**

```
<HTML>
<HEAD>
<TITLE>UserAgent Property Library</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// basic brand determination
function isNav() {
    return (navigator.appName == "Netscape")
}

function isIE() {
    return (navigator.appName == "Microsoft Internet Explorer")
}

// operating system platforms
function isWindows() {
    return (navigator.appVersion.indexOf("Win") != -1)
}

function isWin95NT() {
    return (isWindows() && (navigator.appVersion.indexOf("Win16") == -1 &&
        navigator.appVersion.indexOf("Windows 3.1") == -1))
}

function isMac() {
    return (navigator.appVersion.indexOf("Mac") != -1)
}

function isMacPPC() {
    return (isMac() && (navigator.appVersion.indexOf("PPC") != -1 ||
       navigator.appVersion.indexOf("PowerPC") != -1))
}

function isUnix() {
    return (navigator.appVersion.indexOf("X11") != -1)
}

// browser versions
function isGeneration2() {
    return (parseInt(navigator.appVersion) == 2)
}
```

Continued

Listing 28-1 *(continued)*

```
function isGeneration3() {
    return (parseInt(navigator.appVersion) == 3)
}

function isGeneration3Min() {
    return (parseInt(navigator.appVersion.charAt(0)) >= 3)
}
function isNav4_7() {
    return (isNav() && parseFloat(navigator.appVersion) == 4.7)
}

function isMSIE4Min() {
    return (isIE() && navigator.appVersion.indexOf("MSIE") != -1)
}

function isMSIE5_5() {
    return (navigator.appVersion.indexOf("MSIE 5.5") != -1)
}

function isNN6Min() {
    return (isNav() && parseInt(navigator.appVersion) >= 5)
}

// element referencing syntax
function isDocAll() {
    return (document.all) ? true : false
}

function isDocW3C() {
    return (document.getElementById) ? true : false
}

// fill in the blanks
function checkBrowser() {
    var form = document.forms[0]
    form.brandNN.value = isNav()
    form.brandIE.value = isIE()
    form.win.value = isWindows()
    form.win32.value = isWin95NT()
    form.mac.value = isMac()
    form.ppc.value = isMacPPC()
    form.unix.value = isUnix()
    form.ver3Only.value = isGeneration3()
    form.ver3Up.value = isGeneration3Min()
    form.Nav4_7.value = isNav4_7()
    form.Nav6Up.value = isNN6Min()
    form.MSIE4.value = isMSIE4Min()
    form.MSIE5_5.value = isMSIE5_5()
```

```
    form.doc_all.value = isDocAll()
    form.doc_w3c.value = isDocW3C()
}
</SCRIPT>
</HEAD>

<BODY onLoad="checkBrowser()">
<H1>About This Browser</H1>
<FORM>
<H2>Brand</H2>
Netscape Navigator:<INPUT TYPE="text" NAME="brandNN" SIZE=5>
Internet Explorer:<INPUT TYPE="text" NAME="brandIE" SIZE=5>
<HR>
<H2>Browser Version</H2>
3.0x Only (any brand):<INPUT TYPE="text" NAME="ver3Only" SIZE=5><P>
3 or Later (any brand): <INPUT TYPE="text" NAME="ver3Up" SIZE=5><P>
Navigator 4.7: <INPUT TYPE="text" NAME="Nav4_7" SIZE=5><P>
Navigator 6+: <INPUT TYPE="text" NAME="Nav6Up" SIZE=5><P>
MSIE 4+: <INPUT TYPE="text" NAME="MSIE4" SIZE=5><P>
MSIE 5.5:<INPUT TYPE="text" NAME="MSIE5_5" SIZE=5><P>
<HR>
<H2>OS Platform</H2>
Windows: <INPUT TYPE="text" NAME="win" SIZE=5>
Windows 95/98/2000/NT: <INPUT TYPE="text" NAME="win32" SIZE=5><P>
Macintosh: <INPUT TYPE="text" NAME="mac" SIZE=5>
Mac PowerPC: <INPUT TYPE="text" NAME="ppc" SIZE=5><P>
Unix: <INPUT TYPE="text" NAME="unix" SIZE=5><P>
<HR>
<H2>Element Referencing Style</H2>
Use <TT>document.all</TT>: <INPUT TYPE="text" NAME="doc_all" SIZE=5><P>
Use <TT>document.getElementById()</TT>: <INPUT TYPE="text" NAME="doc_w3c"
SIZE=5><P>
</FORM>
</BODY>
</HTML>
```

Sometimes you may need to use more than one of these functions together. For example, if you want to create a special situation for the `window.open()` bug that afflicts UNIX and Macintosh versions of Navigator 2, then you have to put your Boolean operator logic powers to work to construct a fuller examination of the browser:

```
function isWindowBuggy() {
    return (isGeneration2() && (isMac() || isUnix()))
}
```

You can see many more examples of browser sniffing, including more details about handling AOL browsers, in an article by Eric Krock at: `http://developer.netscape.com:80/docs/examples/javascript/browser_type.html`.

appMinorVersion

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to examine the two related version properties of your IE browser(s). Type the following two statements into the top text box and observe the results:

```
navigator.appVersion
navigator.minorAppVersion
```

There is a good chance that the values returned are not related to the browser version number shown after `MSIE` in the `appVersion` value.

cookieEnabled

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator to see the value of the `navigator.cookieEnabled` property on your browsers. Enter the following statement into the top text box:

```
navigator.cookieEnabled
```

Feel free to change the cookie preferences setting temporarily to see the new value of the property. You do not have to relaunch the browser for the new setting to take effect.

cpuClass

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see how IE reports the `cpuClass` of your PC. Enter the following statement into the top text box:

```
navigator.cpuClass
```

mimeTypes

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			(✓)	(✓)	(✓)

Example

For examples of the `mimeTypes` property and details about using the mimeType object, see the discussion of this object later in the chapter. A number of simple examples showing how to use this property to see whether the `navigator` object has a particular MIME type do not go far enough in determining whether a plug-in is installed and enabled to play the incoming data.

onLine

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see the online state of your IE browsers. Enter the following statement into the top text box:

```
navigator.onLine
```

Verify your browsing mode by checking the Work Offline choice in the File menu. If it is checked, the `onLine` property should return `false`.

oscpu

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) with NN6 to see what your client machine reports to you by entering the following statement into the top text box:

```
navigator.oscpu
```

platform

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see what your computer reports as its operating system. Enter the following statement into the top text box:

```
navigator.platform
```

product
productSub
vendor
vendorSub

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) on your copy of NN6 to see the values returned for these four properties. Enter each of the following statements into the top text box of the page and see the values for each in the Results box:

```
navigator.product
navigator.productSub
navigator.vendor
navigator.vendorSub
```

Also check the value of the `navigator.userAgent` property to see how many of these four property values are revealed in the `userAgent` property.

systemLanguage
userLanguage

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) with your IE4+ browser to compare the values of the three language-related properties running on your computer. Enter each of the following statements into the top text box:

```
navigator.browserLanguage
navigator.systemLanguage
navigator.userLanguage
```

Don't be surprised if all three properties return the same value.

Methods

preference(*name* [, *val*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

The page in Listing 28-2 displays checkboxes for several preference settings, plus one text box to show a preference setting value for the size of the browser's disk cache. To run this script without signing the scripts, turn on codebase principals as directed in Chapter 46 of the *JavaScript Bible*. (The listing file on the CD-ROM does not employ signed scripts.)

One function reads all the preferences and sets the form control values accordingly. Another function sets a preference when you click its checkbox. Because of the interaction among three of the cookie settings, it is easier to have the script rerun the `showPreferences()` function after each setting rather than you trying to manually control the properties of the three checkboxes. Rerunning that function also helps verify that you set the preference.

Listing 28-2: **Reading and Writing Browser Preferences**

```
<HTML>
<HEAD>
<TITLE>Reading/Writing Browser Preferences</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function setPreference(pref, value) {
    netscape.security.PrivilegeManager.enablePrivilege(
       "UniversalPreferencesWrite")
    navigator.preference(pref, value)
    netscape.security.PrivilegeManager.revertPrivilege(
       "UniversalPreferencesWrite")
    showPreferences()
}
```

Continued

Listing 28-2 *(continued)*

```
function showPreferences() {
    var form = document.forms[0]
    netscape.security.PrivilegeManager.enablePrivilege(
        "UniversalPreferencesRead")
    form.imgLoad.checked = navigator.preference("general.always_load_images")
    form.cacheSize.value = navigator.preference("browser.cache.disk_cache_size")
    form.ssEnable.checked = navigator.preference("browser.enable_style_sheets")
    form.autoIEnable.checked = navigator.preference("autoupdate.enabled")
    var cookieSetting = navigator.preference("network.cookie.cookieBehavior")
    for (var i = 0; i < 3; i++) {
        form.elements["cookie" + i].checked = (i == cookieSetting) ? true :
false
    }
    form.cookieWarn.checked =
navigator.preference("network.cookie.warnAboutCookies")
    netscape.security.PrivilegeManager.revertPrivilege(
        "UniversalPreferencesRead")
}
</SCRIPT>
</HEAD>

<BODY onLoad="showPreferences()">
<H1>Browser Preferences Settings Sampler</H1>
<HR>
<FORM>
<INPUT TYPE="checkbox" NAME="imgLoad"
onClick="setPreference('general.always_load_images',this.checked)">
Automatically Load Images<BR>
<INPUT TYPE="checkbox" NAME="ssEnable"
onClick="setPreference('browser.enable_style_sheets',this.checked)">
Style Sheets Enabled<BR>
<INPUT TYPE="checkbox" NAME="autoIEnable"
onClick="setPreference('autoupdate.enabled',this.checked)">
AutoInstall Enabled<BR>
<INPUT TYPE="checkbox" NAME="cookie0"
onClick="setPreference('network.cookie.cookieBehavior',0)">
Accept All Cookies<BR>
<INPUT TYPE="checkbox" NAME="cookie1"
onClick="setPreference('network.cookie.cookieBehavior',1)">
Accept Only Cookies Sent Back to Server<BR>
<INPUT TYPE="checkbox" NAME="cookie2"
onClick="setPreference('network.cookie.cookieBehavior',2)">
Disable Cookies<BR>
<INPUT TYPE="checkbox" NAME="cookieWarn"
onClick="setPreference('network.cookie.warnAboutCookies',this.checked)">
Warn Before Accepting Cookies<BR>
Disk cache is <INPUT TYPE="text" NAME="cacheSize" SIZE=10> KB <BR>
</FORM>
</BODY>
</HTML>
```

screen Object

Properties

availLeft
availTop

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					

Example

If you are a Windows user, you can experiment with these NN4+ properties via The Evaluator (Chapter 13 in the *JavaScript Bible*). With the taskbar at the bottom of the screen, enter these two statements into the top text box:

```
screen.availLeft
screen.availTop
```

Next, drag the taskbar to the top of the screen and try both statements again. Now, drag the taskbar to the left edge of the screen and try the statements once more.

userProfile Object

Methods

addReadRequest("*attributeName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 28-4 in Chapter 28 in the *JavaScript Bible* for an example of the `addReadRequest()` method in action. You can also invoke it from the top text box in The Evaluator (Chapter 13 in the *JavaScript Bible*). For example, enter the following statement to queue one request:

```
navigator.userProfile.addReadRequest("vCard.LastName")
```

To continue the process, see examples for `doReadRequest()` and `getAttribute()` later in this chapter.

doReadRequest(*reasonCode*, *identification*[, *domain*[, *path*[, *expiration*]]])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 28-4 in the *JavaScript Bible* for an example of the `doReadRequest()` method in action. If you entered the `addReadRequest()` example for The Evaluator earlier in this chapter, you can now bring up the permissions dialog box (if you have a user profile for your version of Windows) by entering the following statement into the top text box:

```
navigator.userProfile.doReadRequest(1, "Just me!")
```

getAttribute("*attributeName*")

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

See Listing 28-4 in Chapter 28 in the *JavaScript Bible* for an example of the `getAttribute()` method in action. Also, if you followed The Evaluator examples for this object, you can now extract the desired information (provided it is in your user profile). Enter the following statement into the top text box:

```
navigator.userProfile.getAttribute("vCard.LastName")
```

✦ ✦ ✦

CHAPTER 13

Event Objects (Chapter 29)

In This Chapter

Uncovering the coordinates and target element of a mouse event

Intercepting keyboard events

Observing event propagation in different event object models

As earlier generations of scriptable browsers fade from the installed base, the event models of newer browsers become that much more important to scripters. Although cross-browser developers must concern themselves with the incompatibilities of as many as three distinct event models (NN4, IE4+, and W3C DOM used in NN6), scripts increasingly rely on information conveyed by the event object to know where the event came from.

The importance of event object properties is clear when you see how modern DOMs bind events to objects. Although the "old-fashioned" event handler attribute inside an element tag still works, the prescribed ways to bind events to elements simply assign a function reference to an event type belonging to the event. The significance of this approach is that event handlers no longer receive custom parameters, such as references to the element that used to be passed via the `this` operator. It becomes the job of the function to inspect the event object property that contains a reference to the target of the event.

Fortunately for scripters, the event object model (regardless of which ones you need to support) endows each event object with a list of valuable properties that enhance what event handler functions can do. In addition to character key and mouse button data, you can uncover the coordinates of a mouse event, the condition of modifier keys, and even a reference to the object from which the cursor has just rolled (or where it went after leaving the bounds of the current object).

The code examples in this chapter are grouped by the event object model family. This means that the examples are written to work only within the associated DOM. For cross-browser handling of event objects, see the rest of the discussion in Chapter 29 of the *JavaScript Bible*. But use the examples here to fully understand the meaning of each event object's properties and (in NN6) methods. Where possible, the listings that demonstrate parallel properties in multiple object models look and behave the same to the user; the differences are in the code. As an exercise for the inquisitive, you could write a single-page version that combines syntax from multiple event objects models. Listings 29-17 and 29-22 would be good places to start.

Examples Highlights

- ✦ No fewer than four pairs of coordinate value properties arrive with the IE4+ event object. Listing 29-14 helps you understand what each pair of values represent with respect to regular body elements as well as positioned elements. Follow the suggested steps to experience the meaning of the properties in a variety of contexts.
- ✦ Load Listing 29-16 to see keyboard character data for all three keyboard events. Again, follow the suggested steps to understand important differences among keyboard event types and also different kinds of keys (characters versus non-characters).
- ✦ Listing 29-17 demonstrates how to derive a reference to the element that receives the event in the IE4+ event model.
- ✦ NN6 keyboard events get a workout in Listing 29-18, particularly the way the character and key codes reveal important details for different keyboard event types.
- ✦ All four pairs of event coordinate properties for NN6 are reported when you run Listing 29-19 and click on different elements.
- ✦ The important concepts associated with the NN6 event object's `currentTarget` and `eventPhase` properties are demonstrated in Listing 29-20. Be prepared to spend time with the page and the source code to understand how events propagate through the element hierarchy.
- ✦ Listing 29-23 uses the NN6 `event.timeStamp` property to calculate the instantaneous typing speed within a text field.

NN4 event Object

Properties

`data`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

The page in Listing 29-12 contains little more than a TEXTAREA in which the URLs of dragged items are listed. To run this script without signing the scripts, turn on codebase principals, as directed in Chapter 46 of the *JavaScript Bible*.

To experiment with this listing, load the page and drag any desktop icons that represent files, applications, or folders to the window. Select multiple items and drag them all at once. Because the `onDragDrop` event handler evaluates to `return false`, the files are not loaded into the window. If you want merely to look at the

URL and allow only some to process, you would generate an `if...else` construction to return `true` or `false` to the event handler as needed. A value of `return true` allows the normal processing of the `DragDrop` event to take place after your event handler function has completed its processing.

Listing 29-12: Obtaining URLs of a DragDrop Event's data Property

```
<HTML>
<HEAD>
<TITLE>Drag and Drop</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function handleDrag(evt) {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead")
    var URLArray = evt.data
    netscape.security.PrivilegeManager.disablePrivilege("UniversalBrowserRead")
    if (URLArray) {
        document.forms[0].output.value = URLArray.join("\n")
    } else {
        document.forms[0].output.value = "Nothing found."
    }
    return false
}
</SCRIPT>
</HEAD>
<BODY onDragDrop="return handleDrag(event)">
<B>Drag a URL to this window (NN4 only).</B>
<HR>
<FORM>
URLs:<BR>
<TEXTAREA NAME="output" COLS=70 ROWS=4></TEXTAREA><BR>
<INPUT TYPE="reset">
</FORM>
</BODY>
</HTML>
```

layerX
layerY
pageX
pageY
screenX
screenY

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

You can see the effects of the coordinate systems and associated properties with the page in Listing 29-13. Part of the page contains a three-field readout of the layer-, page-, and screen-level properties. Two clickable objects are provided so that you can see the differences between an object not in any layer and an object residing within a layer. The object not confined by a layer has its layer and page coordinates the same in the event object properties.

Additional readouts display the event object coordinates for resizing and moving a window. If you maximize the window under Windows, the Navigator browser's top-left corner is actually out of sight, four pixels up and to the left. That's why the `screenX` and `screenY` values are both `-4`.

Listing 29-13: **NN4 Event Coordinate Properties**

```
<HTML>
<HEAD>
<TITLE>X and Y Event Properties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function checkCoords(evt) {
    var form = document.forms[0]
    form.layerCoords.value = evt.layerX + "," + evt.layerY
    form.pageCoords.value = evt.pageX + "," + evt.pageY
    form.screenCoords.value = evt.screenX + "," + evt.screenY
    return false
}
function checkSize(evt) {
    document.forms[0].resizeCoords.value = evt.layerX + "," + evt.layerY
}
function checkLoc(evt) {
    document.forms[0].moveCoords.value = evt.screenX + "," + evt.screenY
}
</SCRIPT>
</HEAD>
<BODY onResize="checkSize(event)" onMove="checkLoc(event)">
<H1>X and Y Event Properties (NN4)</H1>
<HR>
<P>Click on the button and in the layer/image to see the coordinate values for
the event object.</P>
<FORM NAME="output">
<TABLE>
<TR><TD COLSPAN=2>Mouse Event Coordinates:</TD></TR>
<TR><TD ALIGN="right">layerX, layerY:</TD><TD><INPUT TYPE="text"
NAME="layerCoords" SIZE=10></TD></TR>
<TR><TD ALIGN="right">pageX, pageY:</TD><TD><INPUT TYPE="text" NAME="pageCoords"
SIZE=10></TD></TR>
<TR><TD ALIGN="right">screenX, screenY:</TD><TD><INPUT TYPE="text"
NAME="screenCoords" SIZE=10></TD></TR>
<TR><TD ALIGN="right"><INPUT TYPE="button" VALUE="Click Here"
onMouseDown="checkCoords(event)"></TD></TR>
<TR><TD COLSPAN=2><HR></TD></TR>
```

```
<TR><TD COLSPAN=2>Window Resize Coordinates:</TD></TR>
<TR><TD ALIGN="right">layerX, layerY:</TD><TD><INPUT TYPE="text"
NAME="resizeCoords" SIZE=10></TD></TR>
<TR><TD COLSPAN=2><HR></TD></TR>
<TR><TD COLSPAN=2>Window Move Coordinates:</TD></TR>
<TR><TD ALIGN="right">screenX, screenY:</TD><TD><INPUT TYPE="text"
NAME="moveCoords" SIZE=10></TD></TR>
</TABLE>
</FORM>
<LAYER NAME="display" BGCOLOR="coral" TOP=140 LEFT=300  HEIGHT=250 WIDTH=330>
<A HREF="javascript:void(0)" onClick="return checkCoords(event)">
<IMG SRC="nile.gif" WIDTH=320 HEIGHT=240" BORDER=0></A>
</LAYER>
</BODY>
</HTML>
```

IE4+ event Object

Properties

clientX
clientY
offsetX
offsetY
screenX
screenY
x
y

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 29-14 provides readings of all event coordinate properties in an interactive way. An `onMouseDown` event handler triggers all event handling, and you can click the mouse anywhere on the page to see what happens. You see the tag of the element targeted by the mouse event to help you visualize how some of the coordinate properties are determined. An image is encased inside a positioned DIV element to help you see what happens to some of the properties when the event is targeted inside a positioned element.

Listing 29-14: **IE4+ Event Coordinate Properties**

```
<HTML>
<HEAD>
<TITLE>X and Y Event Properties (IE4+)</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function checkCoords() {
    var form = document.forms[0]
    form.srcElemTag.value = "<" + event.srcElement.tagName + ">"
    form.clientCoords.value = event.clientX + "," + event.clientY
    form.pageCoords.value = (event.clientX + document.body.scrollLeft) +
        "," + (event.clientY + document.body.scrollTop)
    form.offsetCoords.value = event.offsetX + "," + event.offsetY
    form.screenCoords.value = event.screenX + "," + event.screenY
    form.xyCoords.value = event.x + "," + event.y
    form.parElem.value = "<" + event.srcElement.offsetParent.tagName + ">"
    return false
}
function handleSize() {
    document.forms[0].resizeCoords.value = event.clientX + "," + event.clientY
}
</SCRIPT>
</HEAD>
<BODY onMouseDown="checkCoords()" onResize="handleSize()">
<H1>X and Y Event Properties (IE4+)</H1>
<HR>
<P>Click on the button and in the DIV/image to see the coordinate values for the
event object.</P>
<FORM NAME="output">
<TABLE>
<TR><TD COLSPAN=2>IE Mouse Event Coordinates:</TD></TR>
<TR><TD ALIGN="right">srcElement:</TD><TD><INPUT TYPE="text" NAME="srcElemTag"
SIZE=10></TD></TR>
<TR><TD ALIGN="right">clientX, clientY:</TD><TD><INPUT TYPE="text"
NAME="clientCoords" SIZE=10></TD>
<TD ALIGN="right">...With scrolling:</TD><TD><INPUT TYPE="text"
NAME="pageCoords" SIZE=10></TD></TR>
<TR><TD ALIGN="right">offsetX, offsetY:</TD><TD><INPUT TYPE="text"
NAME="offsetCoords" SIZE=10></TD></TR>
<TR><TD ALIGN="right">screenX, screenY:</TD><TD><INPUT TYPE="text"
NAME="screenCoords" SIZE=10></TD></TR>
<TR><TD ALIGN="right">x, y:</TD><TD><INPUT TYPE="text" NAME="xyCoords"
SIZE=10></TD>
<TD ALIGN="right">...Relative to:</TD><TD><INPUT TYPE="text" NAME="parElem"
SIZE=10></TD></TR>
<TR><TD ALIGN="right"><INPUT TYPE="button" VALUE="Click Here"></TD></TR>
<TR><TD COLSPAN=2><HR></TD></TR>
<TR><TD COLSPAN=2>Window Resize Coordinates:</TD></TR>
<TR><TD ALIGN="right">clientX, clientY:</TD><TD><INPUT TYPE="text"
NAME="resizeCoords" SIZE=10></TD></TR>
</TABLE>
```

```
</FORM>
<DIV ID="display" STYLE="position:relative; left:100">
<IMG SRC="nile.gif" WIDTH=320 HEIGHT=240" BORDER=0>
</DIV>
</BODY>
</HTML>
```

Here are some tasks to try with the page that loads from Listing 29-14 to help you understand the relationships among the various pairs of coordinate properties:

1. Click the dot above the "i" on the "Click Here" button label. The target element is the button (INPUT) element, whose `offsetParent` is a table cell element. The `offsetY` value is very low because you are near the top of the element's own coordinate space. The client coordinates (and `x` and `y`), however, are relative to the viewable area in the window. If your browser window is maximized in Windows, the `screenX` and `clientX` values will be the same; the difference between `screenY` and `clientY` is the height of all the window chrome above the content region. With the window not scrolled at all, the client coordinates are the same with and without scrolling taken into account.
2. Jot down the various coordinate values and then scroll the page down slightly (clicking the scrollbar fires an event) and click the dot on the button again. The `clientY` value shrinks because the page has moved upward relative to the viewable area, making the measure between the top of the area smaller with respect to the button. The Windows version does the right thing with the offset properties, by continuing to return values relative to the element's own coordinate space; the Mac, unfortunately, subtracts the scrolled amount from the offset properties.
3. Click the large image. The client properties perform as expected for both Windows and Mac, as do the screen properties. For Windows, the `x` and `y` properties correctly return the event coordinates relative to the IMG element's `offsetParent`, which is the DIV element that surrounds it. Note, however, that the browser "sees" the DIV as starting 10 pixels to the left of the image. In IE5.5/Windows, you can click within those ten transparent pixels to the left of the image to click the DIV element. This padding is inserted automatically and impacts the coordinates of the `x` and `y` properties. A more reliable measure of the event inside the image is the offset properties. The same is true in the Macintosh version, as long as the page isn't scrolled, in which case the scroll, just as in Step 2, affects the values above.
4. Click the top HR element under the heading. It may take a couple of tries to actually hit the element (you've made it when the HR element shows up in the `srcElement` box). This is to reinforce the way the client properties provide coordinates within the element itself (again, accept on the Mac when the page is scrolled). Clicking at the very left end of the rule, you eventually find the 0,0 coordinate.

Finally, if you are a Windows user, here are two examples to try to see some of the unexpected behavior of coordinate properties.

1. With the page not scrolled, click anywhere along the right side of the page, away from any text so that the BODY element is `srcElement`. Because the BODY element theoretically fills the entire content region of the browser window, all coordinate pairs except for the screen coordinates should be the same. But offset properties are two pixels less than all the others. By and large, this difference won't matter in your scripts, but you should be aware of this potential discrepancy if precise positioning is important. For inexplicable reasons, the offset properties are measured in a space that is inset two pixels from the left and top of the window. This is not the case in the Macintosh version, where all value pairs are the same from the BODY perspective.
2. Click the text of the H1 or P elements (just above and below the long horizontal rule at the top of the page). In theory, the offset properties should be relative to the rectangles occupied by these elements (they're block elements, after all). But instead, they're measured in the same space as the client properties (plus the two pixels). This unexpected behavior doesn't have anything to do with the cursor being a text cursor, because if you click inside any of the text box elements, their offset properties are properly relative to their own rectangles. This problem does not afflict the Macintosh version.

You can see further examples of important event coordinate properties in action in the discussion of dragging elements around the IE page in Chapter 31 of the *JavaScript Bible*.

fromElement
toElement

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 29-15 provides an example of how the `fromElement` and `toElement` properties can reveal the life of the cursor action before and after it rolls into an element. When you roll the cursor to the center box (a table cell), its `onMouseOver` event handler displays the text from the table cell from which the cursor arrived. In Figure 13-1, for example, the user has just rolled the cursor into the center box from the West box. If the cursor comes in from one of the corners (not easy to do), a different message is displayed.

Listing 29-15: Using the toElement and fromElement Properties

```
<HTML>
<HEAD>
<TITLE>fromElement and toElement Properties</TITLE>
<STYLE TYPE="text/CSS">
.direction {background-color:#00FFFF; width:100; height:50; text-align:center}
#main {background-color:#FF6666; text-align:center}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function showArrival() {
    var direction = (event.fromElement.innerText) ? event.fromElement.innerText
:
    "parts unknown"
    status = "Arrived from: " + direction
}
function showDeparture() {
    var direction = (event.toElement.innerText) ? event.toElement.innerText :
    "parts unknown"
    status = "Departed to: " + direction
}
</SCRIPT>
</HEAD>
<BODY>
<H1>fromElement and toElement Properties</H1>
<HR>
<P>Roll the mouse to the center box and look for arrival information
in the status bar. Roll the mouse away from the center box and look for
departure information in the status bar.</P>

<TABLE CELLSPACING=0 CELLPADDING=5>
<TR><TD></TD><TD CLASS="direction">North</TD><TD></TD></TR>
<TR><TD CLASS="direction">West</TD>
<TD ID="main" onMouseOver="showArrival()" onMouseOut="showDeparture()">Roll</TD>
<TD CLASS="direction">East</TD></TR>
<TR><TD></TD><TD CLASS="direction">South</TD><TD></TD></TR>
</TABLE>
</BODY>
</HTML>
```

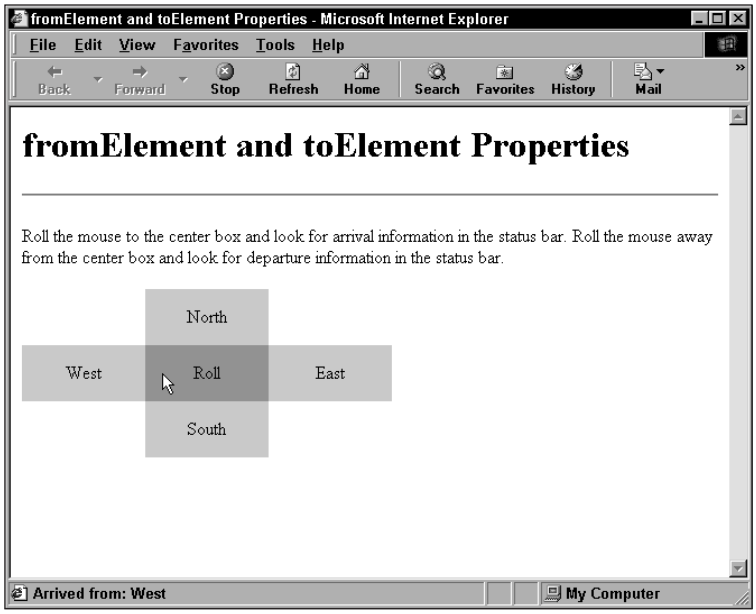


Figure 13-1: onMouseOver event object knows whence the pointer came.

This is a good example to experiment with in the browser, because it also reveals a potential limitation. The element registered as the `toElement` or `fromElement` must fire a mouse event to register itself with the browser. If not, the next element in the sequence that registers itself is the one acknowledged by these properties. For example, if you roll the mouse into the center box and then extremely quickly roll the cursor to the bottom of the page, you may bypass the South box entirely. The text that appears in the statusbar is actually the inner text of the BODY element, which is the element that caught the first mouse event to register itself as the `toElement` for the center table cell.

keyCode

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Listing 29-16 provides an additional play area to view the `keyCode` property for all three keyboard events while you type into a TEXTAREA. You can use this page later as an authoring tool to grab the precise codes for keyboard keys you may not be familiar with.

Listing 29-16: **Displaying keyCode Property Values**

```
<HTML>
<HEAD>
<TITLE>keyCode Property</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function showCode(which) {
    document.forms[0].elements[which].value = event.keyCode
}
function clearEm() {
    for (var i = 1; i < document.forms[0].elements.length; i++) {
        document.forms[0].elements[i].value = ""
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>keyCode Property</H1>
<HR>
<P></P>
<FORM>
<P>
<TEXTAREA NAME="scratchpad" COLS="40" ROWS="5" WRAP="hard"
onKeyDown="clearEm(); showCode('down')" onKeyUp="showCode('up')"
onKeyPress="showCode('press')"></TEXTAREA>
</P>
<TABLE CELLPADDING="5">
<TR><TH>Event</TH><TH>event.keyCode</TH></TR>
<TR><TD>onKeyDown:</TD><TD><INPUT TYPE="text" NAME="down" SIZE="3"></TD></TR>
<TR><TD>onKeyPress:</TD><TD><INPUT TYPE="text" NAME="press" SIZE="3"></TD></TR>
<TR><TD>onKeyUp:</TD><TD><INPUT TYPE="text" NAME="up" SIZE="3"></TD></TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

The following are some specific tasks to try with the page to examine key codes (if you are not using a browser set for English and a Latin-based keyboard, your results may vary):

1. Enter a lowercase letter "a". Notice how the `onKeyPress` event handler shows the code to be 97, which is the Unicode (and ASCII) value for the first of the lowercase letters of the Latin alphabet. But the other two events record just the key's code: 65.
2. Type an uppercase "A" via the Shift key. If you watch closely, you see that the Shift key, itself, generates the code 16 for the `onKeyDown` and `onKeyUp` events.

But the character key then shows the value 65 for all three events, because the ASCII value of the uppercase letter happens to match the keyboard key code for that letter.

3. Press and release the Down Arrow key (be sure the cursor still flashes in the TEXTAREA, because that's where the keyboard events are being monitored). As a non-character key, it does not fire an `onKeyPress` event. But it does fire the other events, and assigns 40 as the code for this key.
4. Poke around with other non-character keys. Some may produce dialog boxes or menus, but their key codes are recorded nonetheless. Note that not all keys on a Macintosh keyboard register with IE/Mac.

returnValue

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

You can find several examples of the `returnValue` property at work in Chapter 15 of the *JavaScript Bible* and in Listings 15-30, 33, 36, 37, 38, and 45 in Chapter 1 of this book. Moreover, many of the other examples in Chapter 15 of the *JavaScript Bible* can substitute the `returnValue` property way of canceling the default action if the scripts were to be run exclusively on IE4+.

srcElement

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

As a simplified demonstration of the power of the `srcElement` property, Listing 29-17 has but two event handlers defined for the BODY element, each invoking a single function. The idea is that the `onMouseDown` and `onMouseUp` events will bubble up from whatever their targets are, and the event handler functions will find out which element is the target and modify the color style of that element.

An extra flair is added to the script in that each function also checks the `className` property of the target element. If the `className` is `bold` — a class name shared by three SPAN elements in the paragraph — the style sheet rule for that class is modified so that all items share the same color (see Figure 13-2). Your scripts can do even more in the way of filtering objects that arrive at the functions to perform special operations on certain objects or groups of objects.

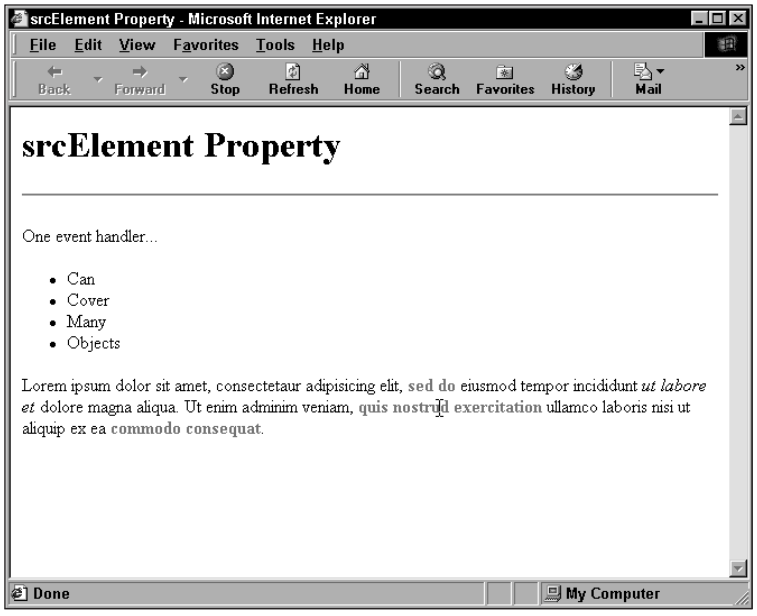


Figure 13-2: Clicking on one SPAN element highlights fellow class members.

Notice that the scripts don't have to know anything about the objects on the page to address each clicked one individually. That's because the `srcElement` property provides all of the specificity needed for acting on the target element.

Listing 29-17: **Using the srcElement property**

```
<HTML>
<HEAD>
<TITLE>srcElement Property</TITLE>
<STYLE TYPE="text/css">
.bold {font-weight:bold}
.ital {font-style:italic}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function highlight() {
    var elem = event.srcElement
    if (elem.className == "bold") {
        document.styleSheets[0].rules[0].style.color = "red"
    } else {
        elem.style.color = "#FFCC00"
    }
}
function restore() {
    var elem = event.srcElement
    if (elem.className == "bold") {
```

Continued

Listing 29-17 *(continued)*

```
        document.styleSheets[0].rules[0].style.color = ""
    } else {
        elem.style.color = ""
    }
}
</SCRIPT>
</HEAD>
<BODY onMouseDown="highlight()" onMouseUp="restore()">
<H1>srcElement Property</H1>
<HR>
<P>One event handler...</P>
<UL>
<LI>Can
<LI>Cover
<LI>Many
<LI>Objects
</UL>
<P>
Lorem ipsum dolor sit amet, consectetaur adipisicing elit,
<SPAN CLASS="bold">sed do </SPAN>eiusmod tempor incididunt
<SPAN CLASS="ital">ut labore et </SPAN>dolore magna aliqua.
Ut enim adminim veniam, <SPAN CLASS="bold">quis nostrud
exercitation </SPAN>ullamco laboris nisi ut aliquip ex ea
<SPAN CLASS="bold">commodo consequat</SPAN>.
</P>
</BODY>
</HTML>
```

type

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see values returned by the `type` property. Enter the following object name into the bottom text box and press Enter/Return:

```
event
```

If necessary, scroll the Results box to view the `type` property, which should read `keypress`. Now click the List Properties button. The type changes to `click`. The reason for these types is that the `event` object whose properties are being shown

here is the event that triggers the function to show the properties. From the text box, an `onKeyPress` event handler triggers that process; from the button, an `onClick` event handler does the job.

NN6+ event Object

charCode
keyCode

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Listing 29-18 provides a play area to view the `charCode` and `keyCode` properties for all three keyboard events while you type into a TEXTAREA. You can use this later as an authoring tool to grab the precise codes for keyboard keys you may not be familiar with.

Listing 29-18: Displaying charCode and keyCode Property Values

```
<HTML>
<HEAD>
<TITLE>charCode and keyCode Properties</TITLE>
<STYLE TYPE="text/css">
TD {text-align:center}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function showCode(which, evt) {
    document.forms[0].elements[which + "Char"].value = evt.charCode
    document.forms[0].elements[which + "Key"].value = evt.keyCode
}
function clearEm() {
    for (var i = 1; i < document.forms[0].elements.length; i++) {
        document.forms[0].elements[i].value = ""
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>charCode and keyCode Properties</H1>
<HR>
<P></P>
<FORM>
<P>
```

Continued

Listing 29-18 *(continued)*

```
<TEXTAREA NAME="scratchpad" COLS="40" ROWS="5" WRAP="hard"
onKeyDown="clearEm(); showCode('down', event)" onKeyUp="showCode('up', event)"
onKeyPress="showCode('press', event)"></TEXTAREA>
</P>
<TABLE CELLPADDING="5">
<TR><TH>Event</TH><TH>event.charCode</TH><TH>event.keyCode</TH></TR>
<TR><TD>onKeyDown:</TD><TD><INPUT TYPE="text" NAME="downChar" SIZE="3"></TD>
<TD><INPUT TYPE="text" NAME="downKey" SIZE="3"></TD></TR>
<TR><TD>onKeyPress:</TD><TD><INPUT TYPE="text" NAME="pressChar" SIZE="3"></TD>
<TD><INPUT TYPE="text" NAME="pressKey" SIZE="3"></TD></TR>
<TR><TD>onKeyUp:</TD><TD><INPUT TYPE="text" NAME="upChar" SIZE="3"></TD>
<TD><INPUT TYPE="text" NAME="upKey" SIZE="3"></TD></TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Here are some specific tasks to try with the page to examine key codes (if you are not using a browser set for English and a Latin-based keyboard, your results may vary):

1. Enter a lowercase letter "a". Notice how the `onKeyPress` event handler shows the `charCode` to be 97, which is the Unicode (and ASCII) value for the first of the lowercase letters of the Latin alphabet. But the other two event types record just the key's code: 65.
2. Type an uppercase "A" via the Shift key. If you watch closely, you see that the Shift key, itself, generates the key code 16 for the `onKeyDown` and `onKeyUp` events. But the character key then shows the value 65 for all three events (until you release the Shift key), because the ASCII value of the uppercase letter happens to match the keyboard key code for that letter.
3. Press and release the Down Arrow key (be sure the cursor still flashes in the TEXTAREA, because that's where the keyboard events are being monitored). As a non-character key, all three events stuff a value into the `keyCode` property, but zero into `charCode`. The `keyCode` value for this key is 40.
4. Poke around with other non-character keys. Some may produce dialog boxes or menus, but their key codes are recorded nonetheless.

```
clientX
clientY
layerX
layerY
pageX
pageY
screenX
screenY
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

You can see the effects of the coordinate systems and associated NN6 properties with the page in Listing 29-19. You can view coordinate values for all four measuring systems, as well as some calculated value. Two clickable objects are provided so that you can see the differences between an object not in any layer and an object residing within a layer (although anything you see is clickable, including text nodes). Figure 13-3 shows the results of a click inside the positioned layer.

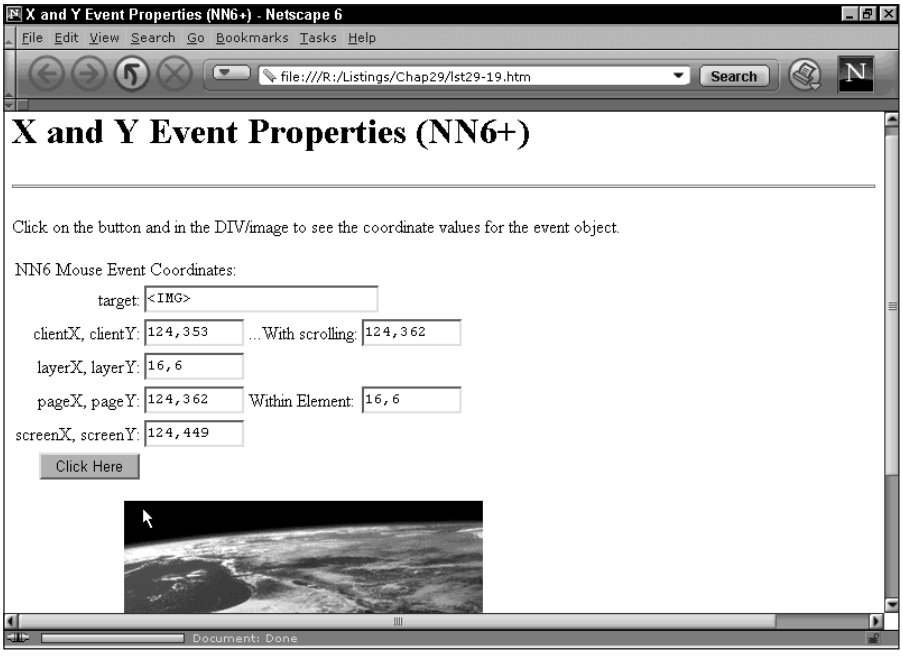


Figure 13-3: NN6 event coordinates for a click inside a positioned element

One of the calculated fields applies window scrolling values to the client coordinates. But, as you will see, these calculated values are the same as the more convenient page coordinates. The other calculated field shows the coordinates relative to the rectangular space of the target element. Notice in the code that if the `nodeType` of the target indicates a text node, that node's parent node (an element) is used for the calculation.

Listing 29-19: **NN6 Event Coordinate Properties**

```
<HTML>
<HEAD>
<TITLE>X and Y Event Properties (NN6+)</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function checkCoords(evt) {
    var form = document.forms["output"]
    var targText, targElem
    if (evt.target.nodeType == 3) {
        targText = "[textnode] inside <" + evt.target.parentNode.tagName + ">"
        targElem = evt.target.parentNode
    } else {
        targText = "<" + evt.target.tagName + ">"
        targElem = evt.target
    }
    form.srcElemTag.value = targText
    form.clientCoords.value = evt.clientX + "," + evt.clientY
    form.clientScrollCoords.value = (evt.clientX + window.scrollX) +
        "," + (evt.clientY + window.scrollY)
    form.layerCoords.value = evt.layerX + "," + evt.layerY
    form.pageCoords.value = evt.pageX + "," + evt.pageY
    form.inElemCoords.value =
    (evt.pageX - targElem.offsetLeft - document.body.offsetLeft) +
        "," + (evt.pageY - targElem.offsetTop - document.body.offsetTop)
    form.screenCoords.value = evt.screenX + "," + evt.screenY
    return false
}
</SCRIPT>
</HEAD>
<BODY onMouseDown="checkCoords(event)">
<H1>X and Y Event Properties (NN6+)</H1>
<HR>
<P>Click on the button and in the DIV/image to see the coordinate values for the
event object.</P>
<FORM NAME="output">
<TABLE>
<TR><TD COLSPAN=2>NN6 Mouse Event Coordinates:</TD></TR>
<TR><TD ALIGN="right">target:</TD>
    <TD COLSPAN=3><INPUT TYPE="text" NAME="srcElemTag" SIZE=25></TD></TR>
<TR><TD ALIGN="right">clientX, clientY:</TD>
    <TD><INPUT TYPE="text" NAME="clientCoords" SIZE=10></TD>
    <TD ALIGN="right">...With scrolling:</TD>
    <TD><INPUT TYPE="text" NAME="clientScrollCoords" SIZE=10></TD></TR>
```

```
<TR><TD ALIGN="right">layerX, layerY:</TD>
    <TD><INPUT TYPE="text" NAME="layerCoords" SIZE=10></TD></TR>
<TR><TD ALIGN="right">pageX, pageY:</TD>
    <TD><INPUT TYPE="text" NAME="pageCoords" SIZE=10></TD>
    <TD ALIGH="right">Within Element:</TD>
    <TD><INPUT TYPE="text" NAME="inElemCoords" SIZE=10></TR>
<TR><TD ALIGN="right">screenX, screenY:</TD>
    <TD><INPUT TYPE="text" NAME="screenCoords" SIZE=10></TD></TR>
<TR><TD ALIGN="right"><INPUT TYPE="button" VALUE="Click Here"></TD></TR>
</TABLE>
</FORM>
<DIV ID="display" STYLE="position:relative; left:100">
<IMG SRC="nile.gif" WIDTH=320 HEIGHT=240" BORDER=0>
</DIV>
</BODY>
</HTML>
```

currentTarget

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Listing 29-20 shows the power of the `currentTarget` property to reveal the element that is processing an event during event propagation. Similar to the code in Listing 29-7, this example is made simpler because it lets the event object's properties do more of the work to reveal the identity of each element that processes the event. Event listeners assigned for various propagation modes are assigned to a variety of nodes in the document. After you click the button, each listener in the propagation chain fires in sequence. The alert dialog shows which node is processing the event. And, as in Listing 29-7, the `eventPhase` property is used to help display the propagation mode in force at the time the event is processed by each node.

Listing 29-20: currentTarget and eventPhase Properties

```
<HTML>
<HEAD>
<TITLE>currentTarget and eventPhase Properties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function init() {
    // using old syntax to assign bubble-type event handlers
    document.onclick = processEvent
    document.body.onclick = processEvent
```

Continued

Listing 29-20 *(continued)*

```
    // turn on click event capture for document and form
    document.addEventListener("click", processEvent, true)
    document.forms[0].addEventListener("click", processEvent, true)
    // set bubble event listener for form
    document.forms[0].addEventListener("click", processEvent, false)
}
function processEvent(evt) {
    var currTargTag, msg
    if (evt.currentTarget.nodeType == 1) {
        currTargTag = "<" + evt.currentTarget.tagName + ">"
    } else {
        currTargTag = evt.currentTarget.nodeName
    }
     msg = "Event is now at the " + currTargTag + " level "
     msg += "(" + getPhase(evt) + ")."
     alert(msg)
}
// reveal event phase of current event object
function getPhase(evt) {
    switch (evt.eventPhase) {
        case 1:
            return "CAPTURING"
            break
        case 2:
            return "AT TARGET"
            break
        case 3:
            return "BUBBLING"
            break
        default:
            return ""
    }
}
</SCRIPT>
</HEAD>
<BODY onLoad="init()">
<H1>currentTarget and eventPhase Properties</H1>
<HR>
<FORM>
<INPUT TYPE="button" VALUE="A Button" NAME="main1"
    onClick="processEvent(event)">
</FORM>
</BODY>
</HTML>
```

You can also click other places on the page. For example, if you click to the right of the button, you will be clicking the FORM element. Event propagation and processing adjusts accordingly. Similarly, if you click the header text, the only event listeners that see the event are in the document and BODY levels.

eventPhase

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

See Listing 29-20 earlier in this chapter for an example of how you can use a `switch` construction to branch function processing based on the event phase of the current event object.

relatedTarget

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Listing 29-21 provides an example of how the `relatedTarget` property can reveal the life of the cursor action before and after it rolls into an element. When you roll the cursor to the center box (a table cell), its `onMouseOver` event handler displays the text from the table cell from which the cursor arrived (the `nodeValue` of the text node inside the table cell). If the cursor comes in from one of the corners (not easy to do), a different message is displayed.

The two functions that report the results employ a bit of filtering to make sure that they process the event object only if the event occurs on an element and if the `relatedTarget` element is anything other than a nested text node of the central table cell element. Because nodes respond to events in NN6, this extra filtering prevents processing whenever the cursor makes the transition from the central TD element to its nested text node.

Listing 29-21: Using the relatedTarget Property

```
<HTML>
<HEAD>
<TITLE>relatedTarget Properties</TITLE>
<STYLE TYPE="text/CSS">
.direction {background-color:#00FFFF; width:100; height:50; text-align:center}
#main {background-color:#FF6666; text-align:center}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function showArrival(evt) {
    if (evt.target.nodeType == 1) {
        if (evt.relatedTarget != evt.target.firstChild) {
```

Continued

Listing 29-21 *(continued)*

```
            var direction = (evt.relatedTarget.firstChild) ?
            evt.relatedTarget.firstChild.nodeValue : "parts unknown"
            status = "Arrived from: " + direction
        }
    }
}
function showDeparture(evt) {
    if (evt.target.nodeType == 1) {
        if (evt.relatedTarget != evt.target.firstChild) {
            var direction = (evt.relatedTarget.firstChild) ?
            evt.relatedTarget.firstChild.nodeValue : "parts unknown"
            status = "Departed to: " + direction
        }
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>relatedTarget Properties</H1>
<HR>
<P>Roll the mouse to the center box and look for arrival information
in the status bar. Roll the mouse away from the center box and look for
departure information in the status bar.</P>

<TABLE CELLSPACING=0 CELLPADDING=5>
<TR><TD></TD><TD CLASS="direction">North</TD><TD></TD></TR>
<TR><TD CLASS="direction">West</TD>
<TD ID="main" onMouseOver="showArrival(event)"
              onMouseOut="showDeparture(event)">Roll</TD>
<TD CLASS="direction">East</TD></TR>
<TR><TD></TD><TD CLASS="direction">South</TD><TD></TD></TR>
</TABLE>
</BODY>
</HTML>
```

target

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

As a simplified demonstration of the power of the `target` property, Listing 29-22 has but two event handlers defined for the BODY element, each invoking a single

function. The idea is that the `onMouseDown` and `onMouseUp` events will bubble up from whatever their targets are, and the event handler functions will find out which element is the target and modify the color style of that element.

An extra flair is added to the script in that each function also checks the `className` property of the target element. If the `className` is `bold` — a class name shared by three SPAN elements in the paragraph — the style sheet rule for that class is modified so that all items share the same color. Your scripts can do even more in the way of filtering objects that arrive at the functions to perform special operations on certain objects or groups of objects.

Notice that the scripts don't have to know anything about the objects on the page to address each clicked one individually. That's because the `target` property provides all of the specificity needed for acting on the target element.

Listing 29-22: **Using the target Property**

```
<HTML>
<HEAD>
<TITLE>target Property</TITLE>
<STYLE TYPE="text/css">
.bold {font-weight:bold}
.ital {font-style:italic}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
function highlight(evt) {
    var elem = (evt.target.nodeType == 3) ? evt.target.parentNode : evt.target
    if (elem.className == "bold") {
        document.styleSheets[0].cssRules[0].style.color = "red"
    } else {
        elem.style.color = "#FFCC00"
    }
}
function restore(evt) {
    var elem = (evt.target.nodeType == 3) ? evt.target.parentNode : evt.target
    if (elem.className == "bold") {
        document.styleSheets[0].cssRules[0].style.color = "black"
    } else {
        elem.style.color = "black"
    }
}
</SCRIPT>
</HEAD>
<BODY onMouseDown="highlight(event)" onMouseUp="restore(event)">
<H1>target Property</H1>
<HR>
<P>One event handler...</P>
<UL>
<LI>Can
<LI>Cover
<LI>Many
<LI>Objects
```

Continued

Listing 29-22 *(continued)*

```
</UL>
<P>
Lorem ipsum dolor sit amet, consectetaur adipisicing elit,
<SPAN CLASS="bold">sed do </SPAN>eiusmod tempor incididunt
<SPAN CLASS="ital">ut labore et </SPAN>dolore magna aliqua.
Ut enim adminim veniam, <SPAN CLASS="bold">quis nostrud
exercitation </SPAN>ullamco laboris nisi ut aliquip ex ea
<SPAN CLASS="bold">commodo consequat</SPAN>.
</P>
</BODY>
</HTML>
```

timeStamp

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Listing 29-23 uses the `timeStamp` property to calculate the instantaneous typing speed when you type into a TEXTAREA (see Figure 13-4). The calculations are pretty raw and work only on intra-keystroke times without any averaging or smoothing that a more sophisticated typing tutor might perform. Calculated values are rounded to the nearest integer.

Listing 29-23: **Using the timeStamp property**

```
<HTML>
<HEAD>
<TITLE>timeStamp Property</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var stamp
function calcSpeed(evt) {
    if (stamp) {
        var gross = evt.timeStamp - stamp
        var wpm = Math.round(6000/gross)
        document.getElementById("wpm").firstChild.nodeValue = wpm + " wpm."
    }
    stamp = evt.timeStamp

}
</SCRIPT>
</HEAD>
```

```
<BODY>
<H1>timeStamp Property</H1>
<HR>
<P>Start typing, and watch your instantaneous typing speed below:</P>
<P>
<TEXTAREA COLS=60 ROWS=10 WRAP="hard" onKeyPress="calcSpeed(event)"></TEXTAREA>
</P>
<P>Typing Speed: <SPAN ID="wpm"> </SPAN></P>
</BODY>
</HTML>
```

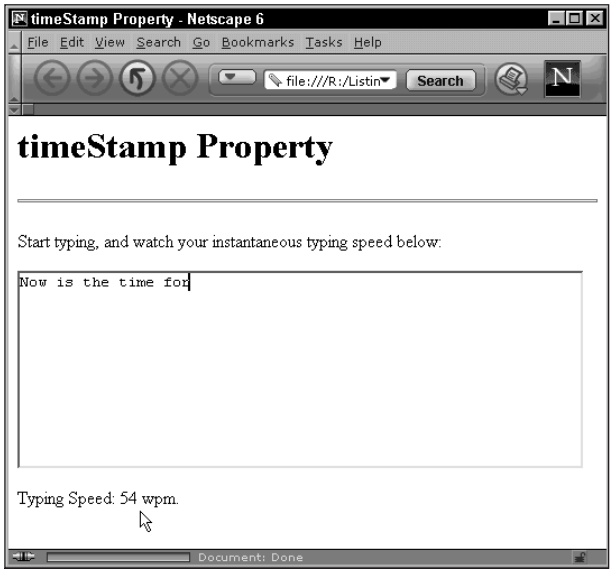


Figure 13-4: The timeStamp property helps calculate typing speed.

✦ ✦ ✦

CHAPTER 14

Style Sheet Objects (Chapter 30)

In This Chapter

Enabling and disabling entire style sheets

Accessing an individual style rule from a style sheet

Adding and deleting style sheet rules

Examples in this chapter focus on the properties and methods of the `styleSheet` object. As described in Chapter 30 of the *JavaScript Bible*, object models that support scriptable style sheets define both the STYLE element object (representing the element created with a `<STYLE>` tag pair) and the more abstract `styleSheet` object. The latter may be created by virtue of a STYLE element or perhaps imported from an external style sheet definition file.

Use the `styleSheet` object to gain access to the details of the rules defined for a given style sheet. Methods of the `styleSheet` object (different syntax for IE4+ and W3C object models) allow dynamic creation or deletion of rules within a style sheet. Properties of the `styleSheet` object (again, different syntax) return arrays of objects representing the style rules contained by the style sheet. The rule objects themselves have properties allowing reading and writing of rule selectors and even individual style attributes within that rule (since a single rule can list multiple style attributes).

Examples Highlights

- Compare examples for the `styleSheet.cssRules` and `styleSheet.rules` properties to see how different browsers provide access to arrays of rule objects.
- You can observe in The Evaluator (Chapter 13 in the *JavaScript Bible*) how the `styleSheet.disabled` property can switch a style sheet on and off dynamically.
- Compare the `styleSheet` object method pairs for inserting and deleting rules to an existing style sheet. The walk-through examples let you follow the same steps for both the IE4+ and NN6 syntaxes.
- The final example in this chapter demonstrates how scripts can modify a single attribute of a style sheet rule.

The syntax in the demonstration is for NN6 and IE5/Mac, but referencing the `cssRules` property provides the same access for the IE4+ object model.

styleSheet Object

Properties

cssRules

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓				(✓)	(✓)

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to look at the `cssRules` property in NN6+ or IE5+/Mac. First, view how many rules are in the first styleSheet object of the page by entering the following statement into the top text box:

```
document.styleSheets[0].cssRules.length
```

Now use the array with an index value to access one of the rule objects to view the rule object's properties list. Enter the following statement into the bottom text box:

```
document.styleSheets[0].cssRules[1]
```

You use this syntax to modify the style details of an individual rule belonging to the styleSheet object.

cssText

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

Example

Use The Evaluator (Chapter 13) to replace the style rules in one blast via the `cssText` property. Begin by examining the value returned from the property for the initially disabled style sheet by entering the following statement into the top text box:

```
document.styleSheets[0].cssText
```

Next, enable the style sheet so that its rules are applied to the document:

```
document.styleSheets[0].disabled = false
```

Finally, enter the following statement into the top text box to overwrite the style sheet with entirely new rules.

```
document.styleSheets[0].cssText = "P {color:red}"
```

Reload the page after you are finished to restore the original state.

disabled

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to toggle between the enabled and disabled state of the first styleSheet object on the page. Enter the following statement into the top text box:

```
document.styleSheets[0].disabled = (!document.styleSheets[0].disabled)
```

The inclusion of the NOT operator (`!`) forces the state to change from `true` to `false` or `false` to `true` with each click of the Evaluate button.

ownerNode

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) with NN6 to inspect the `ownerNode` of the first styleSheet object in the document. Enter the following statement into the top text box:

```
document.styleSheets[0].ownerNode.tagName
```

The returned value is the STYLE element tag name.

owningElement

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in *JavaScript Bible*) with IE4+ to inspect the `owningElement` of the first styleSheet object in the document. Enter the following statement into the top text box:

```
document.styleSheets[0].owningElement.tagName
```

The returned value is the STYLE element tag name.

rules

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) with IE4+ to examine the `rules` property of the first styleSheet object in the page. First, find out how many rules are in the first styleSheet object by entering the following statement into the top text box:

```
document.styleSheets[0].rules.length
```

Next, examine the properties of one of the rules by entering the following statement into the bottom text box:

```
document.styleSheets[0].rules[1]
```

You now see the all the properties that IE4+ exposes for a rule object.

Methods

addRule("*selector*", "*styleSpec*"[, *index*])
removeRule(*index*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility							✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) with IE4+ to add a style sheet rule to the first styleSheet object of the page. First, make sure the style sheet is enabled by entering the following statement into the top text box:

```
document.styleSheets[0].disabled = false
```

Next, append a style that sets the color of the TEXTAREA element:

```
document.styleSheets[0].addRule("TEXTAREA", "color:red")
```

Enter any valid object (such as `document.body`) into the bottom text box to see how the style has been applied to the TEXTAREA element on the page.

Now remove the style, using the index of the last item of the `rules` collection as the index:

```
document.styleSheets[0].removeRule(document.styleSheets[0].rules.length - 1)
```

The text in the TEXTAREA returns to its default color.

deleteRule(*index*)
insertRule("*rule*", *index*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) with NN6+ to add a style sheet rule to the first styleSheet object of the page. First, make sure the style sheet is enabled by entering the following statement into the top text box:

```
document.styleSheets[0].disabled = false
```

Next, append a style that sets the color of the TEXTAREA element:

```
document.styleSheets[0].insertRule("TEXTAREA {color:red}",
document.styleSheets[0].cssRules.length)
```

Enter any valid object (such as `document.body`) into the bottom text box to see how the style has been applied to the TEXTAREA element on the page.

Now remove the style, using the index of the last item of the rules collection as the index:

```
document.styleSheets[0].deleteRule(document.styleSheets[0].cssRules.length - 1)
```

The first release of NN6 processes most, but not all, of the internal actions in response to the `deleteRule()` method. The method returns no value, so the Results box correctly reports `undefined` after evaluating the `deleteRule()` example statement. At the same time, the method has genuinely removed the rule from the styleSheet object (as proven by inspecting the `length` property of the `document.styleSheets[0].cssRules` array). But the browser does not refresh the page display to reflect the removal of the rule.

cssRule and rule Objects

Properties

selectorText

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to examine the `selectorText` property of rules in the first styleSheet object of the page. Enter each of the following statements in the top text box:

```
document.styleSheets[0].rules[0].selectorText
document.styleSheets[0].rules[1].selectorText
```

Compare these values against the source code view for the STYLE element in the page.

style

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to modify a `style` property of one of the styleSheet rules in the page. The syntax shown here is for IE4+, but you can substitute the `cssRules` reference for the `rules` collection reference in NN6 (and IE5/Mac) if you like.

Begin by reloading the page and making sure the style sheet is enabled. Enter the following statement into the top text box:

```
document.styleSheets[0].disabled = false
```

The first rule is for the `myP` element on the page. Change the rule's `font-size` style:

```
document.styleSheets[0].rules[0].style.fontSize = "20pt"
```

Look over the `style` object properties in the discussion of the `style` object later in this chapter and have fun experimenting with different style properties. After you are finished, reload the page to restore the styles to their default states.

✦ ✦ ✦

CHAPTER 15

The NN4 Layer Object (Chapter 31)

✦ ✦ ✦ ✦

In This Chapter

Using NN4-specific syntax for positioned elements

How to move, hide, and show positioned content in NN4

Setting the clipping rectangle of a layer in NN4

✦ ✦ ✦ ✦

Chapter 31 of the *JavaScript Bible* is devoted to positioned objects in all object models. Only Navigator 4 has its own set of dedicated positionable objects: the LAYER and ILAYER element objects. In the IE4+ and W3C DOMs, virtually any renderable element is positionable, although it is common practice to restrict such activity to SPAN and DIV elements. Because properties of the SPAN, DIV, and other HTML element objects are covered in detail in other chapters, Chapter 31 provides the details of the NN4 layer object.

Examples shown here support NN4 layer object details, but the rest of the discussion and code listings in *JavaScript Bible* Chapter 31 go to great lengths to recreate the same behaviors in both the IE4+ and W3C (NN6) object models. This will help those scripters who developed extensively for NN4's Dynamic HTML make the transition to NN6 and its support for Dynamic HTML (which is not much different from that in the IE4+ object model). Obviously, all examples shown below require NN4.

Examples Highlights

- Clipping of layer rectangles is not an easy concept to grasp at first (in any object model). Listing 31-2 provides a workbench to explore the various properties associated with the clipping rectangle. Listing 31-5 demonstrates the relationship between moving a layer and adjusting its clipping rectangle.
- Listing 31-6 is an extensive demonstration of a variety of layer coordinate system properties.
- Most layer object properties are handled in later object models through style sheet property manipulation. Listing 31-8 shows the NN4 layer way of handling a layer's visibility, while Listing 31-9 demonstrates adjusting the stacking order of layers.

✦ Scripts for dragging a layer (with the help of the layer object's move methods) appear in Listing 31-11. Another type of dragging — dragging a corner to resize a layer — takes center stage in Listing 31-12a.

NN4 Layer Object

Properties

above
below
siblingAbove
siblingBelow

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

Listing 31-1 enables you to experiment with just one set of these properties: *layerObject*.above and *layerObject*.below. The page is almost in the form of a laboratory/quiz that enables you to query yourself about the values of these properties for two swappable layers.

Listing 31-1: A Layer Quiz

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function checkAbove(oneLayer) {
    document.forms[0].errors.value = ""
    document.forms[0].output.value = oneLayer.above.name
}
function checkBelow(oneLayer) {
    document.forms[0].errors.value = ""
    document.forms[0].output.value = oneLayer.below.name
}
function swapLayers() {
    if (document.yeller.above) {
        document.yeller.moveAbove(document.greeny)
    } else {
        document.greeny.moveAbove(document.yeller)
    }
}
```

```
function onerror(msg) {
    document.forms[0].output.value = ""
    document.forms[0].errors.value = msg
    return true
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Layer Ordering</H1>
<HR>
<FORM>
Results:<INPUT TYPE="text" NAME="output"><P>
<INPUT TYPE="button" VALUE="Who's ABOVE the Yellow layer?"
onClick="checkAbove(document.yeller)"><BR>
<INPUT TYPE="button" VALUE="Who's BELOW the Yellow layer?"
onClick="checkBelow(document.yeller)"><P>
<INPUT TYPE="button" VALUE="Who's ABOVE the Green layer?"
onClick="checkAbove(document.greeny)"><BR>
<INPUT TYPE="button" VALUE="Who's BELOW the Green layer?"
onClick="checkBelow(document.greeny)"><P>
<INPUT TYPE="button" VALUE="Swap Layers" onCLick="swapLayers()"><P>
If there are any errors caused by missing <BR>
properties, they will appear below:<BR>
<TEXTAREA NAME="errors" COLS=30 ROWS=3 WRAP="virtual"></TEXTAREA>
</FORM>
<LAYER NAME="yeller" BGCOLOR="yellow" TOP=110 LEFT=300 WIDTH=200 HEIGHT=200>
<B>This is just a yellow layer.</B>
</LAYER>
<LAYER NAME="greeny" BGCOLOR="lightgreen" TOP=150 LEFT=340 WIDTH=200 HEIGHT=200>
<B>This is just a green layer.</B>
</LAYER>
</BODY>
</HTML>
```

The page contains two layers: one colored yellow and the other light green. Legends on four buttons ask you to guess whether one layer is above or below the other. For example, if you click the button labeled "Who's ABOVE the Yellow layer?" and the green layer is above it, the name of that green layer appears in the Results field. But if layers are oriented such that the returned value is `null`, the error message (indicating that the nonexistent object doesn't have a `name` property) appears in the error field at the bottom. Another button enables you to swap the order of the layers so you can try your hand at predicting the results based on your knowledge of layers and the `above` and `below` properties. Positioned objects in IE4+ and NN6 have no comparable properties to the four described in this section.

background

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

A simple example (Listing 31-2) defines one layer that features five buttons to change the background image of a second layer. I put the buttons in a layer because I want to make sure the buttons and background layer rectangles align themselves along their top edges on all platforms.

As the second layer loads, I merely assign a gray background color to it and write some reverse (white) text. Most of the images are of the small variety that repeat in the layer. One is a large photograph to demonstrate how images are clipped to the layer's rectangle. Along the way, I hope you also heed the lesson of readability demonstrated by the difficulty of reading text on a wild-looking background. For an example compatible with IE5+ and NN6+, see Listing 31-13.

Listing 31-2: **Setting Layer Backgrounds**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function setBg(URL) {
    document.bgExpo.background.src = URL
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Layer Backgrounds</H1>
<HR>
<LAYER NAME="buttons" TOP=100>
    <FORM>
        <INPUT TYPE="button" VALUE="The Usual"
onClick="setBg('cr_kraft.gif')"><BR>
        <INPUT TYPE="button" VALUE="A Big One" onClick="setBg('arch.gif')"><BR>
        <INPUT TYPE="button" VALUE="Not So Usual"
onClick="setBg('wh86.gif')"><BR>
        <INPUT TYPE="button" VALUE="Decidedly Unusual"
onClick="setBg('sb23.gif')"><BR>
        <INPUT TYPE="button" VALUE="Quick as..."
onClick="setBg('lightnin.gif')"><BR>
    </FORM>
</LAYER>
<LAYER NAME="bgExpo" BGCOLOR="gray" TOP=100 LEFT=250 WIDTH=300 HEIGHT=260>
<B><FONT COLOR="white">Some text, which may or may not read well with the
various backgrounds.</FONT></B>
```

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

bgColor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

You can have some fun with Listing 31-3, which uses a number of layer scripting techniques. The page presents a kind of palette of eight colors, each one created as a small layer (see Figure 15-1). Another, larger layer's `bgColor` property changes as you roll the mouse over any color in the palette.

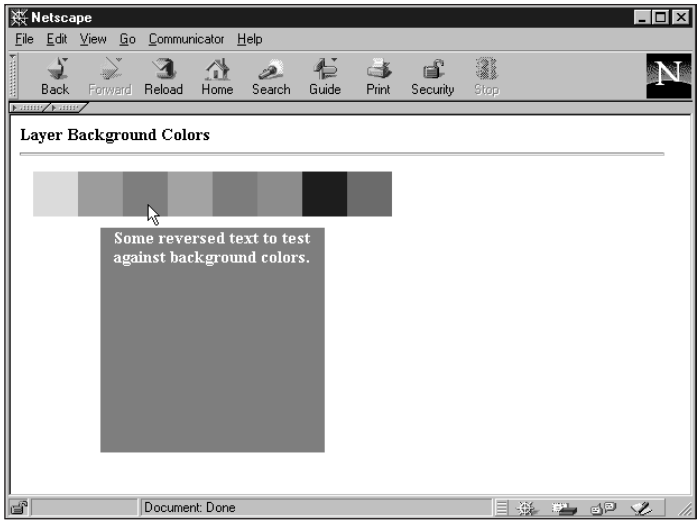


Figure 15-1: Drag the mouse across the palette to change the layer's background color.

To save HTML lines to create those eight color palette layers, I use a script to establish an array of colors and then `document.write()` the `<LAYER>` tags with appropriate attribute settings so the layers all line up in a contiguous row. By predefining a number of variable values for the size of the color layers, I can make all of them larger or smaller with the change of only a few script characters.

The `document` object handles the job of capturing the `mouseOver` events. I turn on the document's `captureEvents()` method such that it traps all `mouseOver`

events and hands them to the `setColor()` function. The `setColor()` function reads the target object's `bgColor` and sets the larger layer's `bgColor` property to the same. If this page had other objects that could receive `mouseOver` events for other purposes, I would use `routeEvents()` to let those events pass on to their intended targets. For the purposes of this example, however, the events need to go no further. Listing 31-14 in the *JavaScript Bible* shows the same functionality working in IE5+ and NN6+.

Listing 31-3: **Layer Background Colors**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function setColor(e) {
    document.display.bgColor = e.target.bgColor
}
document.captureEvents(Event.MOUSEOVER)
document.onmouseover = setColor
</SCRIPT>
</HEAD>
<BODY>
<H1>Layer Background Colors</H1>
<HR>
<SCRIPT LANGUAGE="JavaScript">
var oneLayer
var colorTop = 100
var colorLeft = 20
var colorWidth = 40
var colorHeight = 40
var colorPalette = new
Array("aquamarine","coral","forestgreen","goldenrod","red",
                            "magenta","navy","teal")
for (var i = 0; i < colorPalette.length; i++) {
    oneLayer = "<LAYER NAME=swatch" + i + " TOP=" + colorTop
    oneLayer += " LEFT=" + ((colorWidth * i) + colorLeft)
    oneLayer += " WIDTH=" + colorWidth + " HEIGHT=" + colorHeight
    oneLayer += " BGCOLOR=" + colorPalette[i] + "></LAYER>\n"
    document.write(oneLayer)
}
</SCRIPT>
<LAYER NAME="display" BGCOLOR="gray" TOP=150 LEFT=80 WIDTH=200 HEIGHT=200>
<B><FONT COLOR="white"><CENTER>Some reversed text to test against background
colors.</CENTER></FONT></B>
</LAYER>
</BODY>
</HTML>
```

clip

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

Because of the edge movement behavior of adjustments to *layerObject*.clip properties, Listing 31-4 enables you to experiment with adjustments to each of the six properties. The document loads one layer that you can adjust by entering alternative values into six text fields — one per property. Figure 15-2 shows the page.

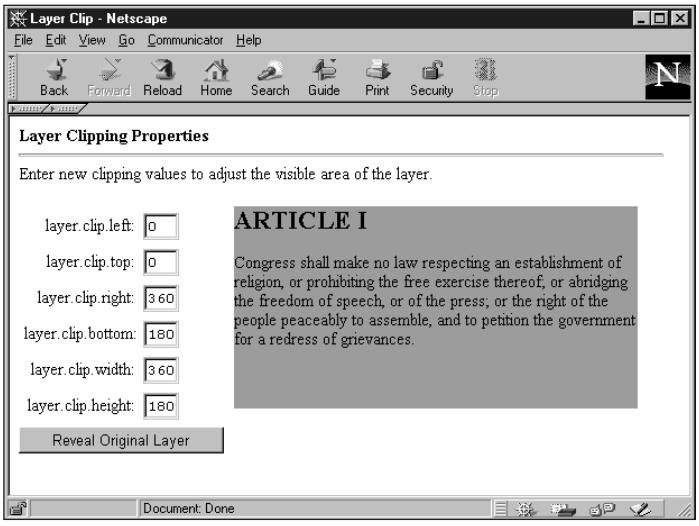


Figure 15-2: Experiment with layer.clip properties.

As you enter values, all properties are updated to show their current values (via the showValues() function). Pay particular attention to the apparent motion of the edge and the effect the change has on at least one other property. For example, a change to the *layerObject*.clip.left value also affects the *layerObject*.clip.width property value.

Listing 31-4: **Adjusting layer.clip Properties**

```
<HTML>
<HEAD>
<TITLE>Layer Clip</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var origLayerWidth = 0
var origLayerHeight = 0
```

Continued

Listing 31-4 *(continued)*

```
function initializeXY() {
    origLayerWidth = document.display.clip.width
    origLayerHeight = document.display.clip.height
    showValues()
}

function setClip(field) {
    var clipVal = parseInt(field.value)
    document.display.clip[field.name] = clipVal
    showValues()
}
function showValues() {
    var form = document.layers[0].document.forms[0]
    var propName
    for (var i = 0; i < form.elements.length; i++) {
        propName = form.elements[i].name
        if (form.elements[i].type == "text") {
            form.elements[i].value = document.display.clip[propName]
        }
    }
}
var intervalID
function revealClip() {
    var midWidth = Math.round(origLayerWidth /2)
    var midHeight = Math.round(origLayerHeight /2)
    document.display.clip.left = midWidth
    document.display.clip.top = midHeight
    document.display.clip.right = midWidth
    document.display.clip.bottom = midHeight
    intervalID = setInterval("stepClip()",1)
}
function stepClip() {
    var widthDone = false
    var heightDone = false
    if (document.display.clip.left > 0) {
        document.display.clip.left += -2
        document.display.clip.right += 2
    } else {
        widthDone = true
    }
    if (document.display.clip.top > 0) {
        document.display.clip.top += -1
        document.display.clip.bottom += 1
    } else {
        heightDone = true
    }
    showValues()
    if (widthDone && heightDone) {
        clearInterval(intervalID)
```

```
    }
}
</SCRIPT>
</HEAD>
<BODY onLoad="initializeXY()">
<H1>Layer Clipping Properties</H1>
<HR>
Enter new clipping values to adjust the visible area of the layer.<P>
<LAYER TOP=130>
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right">layer.clip.left:</TD>
    <TD><INPUT TYPE="text" NAME="left" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.clip.top:</TD>
    <TD><INPUT TYPE="text" NAME="top" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.clip.right:</TD>
    <TD><INPUT TYPE="text" NAME="right" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.clip.bottom:</TD>
    <TD><INPUT TYPE="text" NAME="bottom" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.clip.width:</TD>
    <TD><INPUT TYPE="text" NAME="width" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.clip.height:</TD>
    <TD><INPUT TYPE="text" NAME="height" SIZE=3 onChange="setClip(this)"></TD>
</TR>
</TABLE>
<INPUT TYPE="button" VALUE="Reveal Original Layer" onClick="revealClip()">
</FORM>
</LAYER>
<LAYER NAME="display" BGCOLOR="coral" TOP=130 LEFT=200 WIDTH=360 HEIGHT=180>
<H2>ARTICLE I</H2>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of speech, or of
the press; or the right of the people peaceably to assemble, and to petition the
government for a redress of grievances.
</P>
</LAYER>
</BODY>
</HTML>
```

Listing 31-4 has a lot of other scripting in it to demonstrate a couple of other clip area techniques. After the document loads, the `onLoad` event handler initializes two global variables that represent the starting height and width of the layer as determined by the `clip.height` and `clip.width` properties. Because the `<LAYER>` tag does not specify any `CLIP` attributes, the *`layerObject`*`.clip` region is ensured of being the same as the layer's dimensions at load time.

I preserve the initial values for a somewhat advanced set of functions that act in response to the Reveal Original Layer button. The goal of this button is to temporarily shrink the clipping area to nothing and then expand the clip rectangle gradually from the very center of the layer. The effect is analogous to a zoom-out visual effect.

The clip region shrinks to practically nothing by setting all four edges to the same point midway along the height and width of the layer. The script then uses `setInterval()` to control the animation in `setClip()`. To make the zoom even on both axes, I first make sure that the initial size of the layer is an even ratio: twice as wide as it is tall. Each time through the `setClip()` function, the `clip.left` and `clip.right` values are adjusted in their respective directions by two pixels and `clip.top` and `clip.bottom` are adjusted by one pixel.

To make sure the animation stops when the layer is at its original size, I check whether the `clip.top` and `clip.left` values are their original zero values. If they are, I set a Boolean variable for each side. When both variables indicate that the clip rectangle is its original size, the script cancels the `setInterval()` action. Listing 31-15 in the *JavaScript Bible* demonstrates how to adjust clipping in IE5+ and NN6+ syntax.

left
top

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

To enable you to experiment with manually setting *`layerObject`*`.top` and *`layerObject`*`.left` properties, Listing 31-5 is a modified version of the `layer.clip` example (Listing 31-4). The current example again has the one modifiable layer, but it has only four text fields in which you can enter values. Two fields are for the *`layerObject`*`.left` and *`layerObject`*`.top` properties; the other two are for the *`layerObject`*`.clip.left` and *`layerObject`*`.clip.top` properties. I present both sets of values here to help reinforce the lack of connection between layer and clip location properties in the same layer object. You can find the corresponding syntax for IE5+ and NN6+ in Listing 31-16 of the *JavaScript Bible*.

Listing 31-5: Comparison of Layer and Clip Location Properties

```
<HTML>
<HEAD>
<TITLE>Layer vs. Clip</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setClip(field) {
    var clipVal = parseInt(field.value)
    document.display.clip[field.name] = clipVal
    showValues()
}
function setLayer(field) {
    var layerVal = parseInt(field.value)
    document.display[field.name] = layerVal
    showValues()
}
function showValues() {
    var form = document.layers[0].document.forms[0]
    form.elements[0].value = document.display.left
    form.elements[1].value = document.display.top
    form.elements[2].value = document.display.clip.left
    form.elements[3].value = document.display.clip.top
}
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<B>Layer vs. Clip Location Properties</B>
<HR>
Enter new layer and clipping values to adjust the layer.<P>
<LAYER TOP=80>
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right">layer.left:</TD>
    <TD><INPUT TYPE="text" NAME="left" SIZE=3 onChange="setLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.top:</TD>
    <TD><INPUT TYPE="text" NAME="top" SIZE=3 onChange="setLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.clip.left:</TD>
    <TD><INPUT TYPE="text" NAME="left" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.clip.top:</TD>
    <TD><INPUT TYPE="text" NAME="top" SIZE=3 onChange="setClip(this)"></TD>
</TR>
</TABLE>
</FORM>
```

Continued

Listing 31-5 *(continued)*

```
</LAYER>
<LAYER NAME="display" BGCOLOR="coral" TOP=80 LEFT=200 WIDTH=360 HEIGHT=180>
<H2>ARTICLE I</H2>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of speech, or of
the press; or the right of the people peaceably to assemble, and to petition the
government for a redress of grievances.
</P>
</LAYER>
</BODY>
</HTML>
```

pageX
pageY

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

Listing 31-6 defines one outer layer and one nested inner layer of different colors (see Figure 15-3). The inner layer contains some text content; the outer layer is sized initially to present a colorful border by being below the inner layer and 10 pixels wider and taller.

Two sets of fields display (and enable you to change) the *layerObject*.pageX, *layerObject*.pageY, *layerObject*.left, and *layerObject*.top properties for each of the nested layers. Each set of fields is color-coded to its corresponding layer.

When you change any value, all values are recalculated and displayed in the other fields. For example, the initial pageX position for the outer layer is 200 pixels; for the inner layer, the pageX value is 205 pixels (accounting for the 5-pixel "border" around the inner layer). If you change the outer layer's pageX value to 220, the outer layer moves to the right by 20 pixels, taking the inner layer along for the ride. The layer.pageX value for the inner layer after the move is 225 pixels.

The outer layer values for the pairs of values are always the same no matter what. But for the inner layer, the page values are significantly different from the layer.left and layer.top values because these latter values are measured relative to their containing layer — the outer layer. If you move the outer layer, the inner layer values for *layerObject*.left and *layerObject*.top don't change one iota. Listing 31-17 in the *JavaScript Bible* shows the comparable syntax for IE5+ and NN6+.

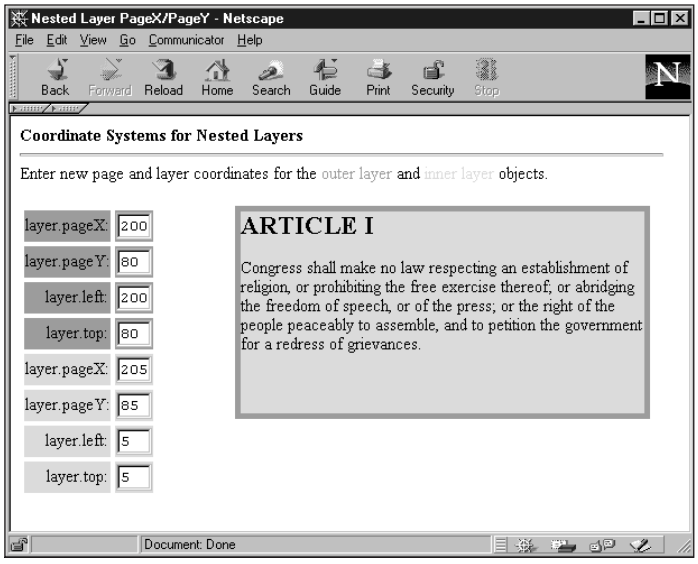


Figure 15-3: Testing the position properties of nested layers

Listing 31-6: **Testing Nested Layer Coordinate Systems**

```
<HTML>
<HEAD>
<TITLE>Nested Layer PageX/PageY</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setOuterPage(field) {
    var layerVal = parseInt(field.value)
    document.outerDisplay[field.name] = layerVal
    showValues()
}
function setOuterLayer(field) {
    var layerVal = parseInt(field.value)
    document.outerDisplay[field.name] = layerVal
    showValues()
}
function setInnerPage(field) {
    var layerVal = parseInt(field.value)
    document.outerDisplay.document.innerDisplay[field.name] = layerVal
    showValues()
}
function setInnerLayer(field) {
    var layerVal = parseInt(field.value)
    document.outerDisplay.document.innerDisplay[field.name] = layerVal
    showValues()
}
```

Continued

Listing 31-6 *(continued)*

```
function showValues() {
    var form = document.layers[0].document.forms[0]
    form.elements[0].value = document.outerDisplay.pageX
    form.elements[1].value = document.outerDisplay.pageY
    form.elements[2].value = document.outerDisplay.left
    form.elements[3].value = document.outerDisplay.top
    form.elements[4].value = document.outerDisplay.document.innerDisplay.pageX
    form.elements[5].value = document.outerDisplay.document.innerDisplay.pageY
    form.elements[6].value = document.outerDisplay.document.innerDisplay.left
    form.elements[7].value = document.outerDisplay.document.innerDisplay.top
}
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<B>Coordinate Systems for Nested Layers</B>
<HR>
Enter new page and layer coordinates for the <FONT COLOR="coral">outer
layer</FONT> and <FONT COLOR="aquamarine">inner layer</FONT> objects.<P>
<LAYER TOP=80>
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right" BGCOLOR="coral">layer.pageX:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="pageX" SIZE=3
        onChange="setOuterPage(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="coral">layer.pageY:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="pageY" SIZE=3
        onChange="setOuterPage(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="coral">layer.left:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="left" SIZE=3
        onChange="setOuterLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="coral">layer.top:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="top" SIZE=3
        onChange="setOuterLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="aquamarine">layer.pageX:</TD>
    <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="pageX" SIZE=3
        onChange="setInnerPage(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="aquamarine">layer.pageY:</TD>
    <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="pageY" SIZE=3
        onChange="setInnerPage(this)"></TD>
</TR>
```

```
<TR>
    <TD ALIGN="right" BGCOLOR="aquamarine">layer.left:</TD>
    <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="left" SIZE=3
        onChange="setInnerLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="aquamarine">layer.top:</TD>
    <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="top" SIZE=3
        onChange="setInnerLayer(this)"></TD>
</TR>
</TABLE>
</FORM>
</LAYER>
<LAYER NAME="outerDisplay" BGCOLOR="coral" TOP=80 LEFT=200 WIDTH=370 HEIGHT=190>
<LAYER NAME="innerDisplay" BGCOLOR="aquamarine" TOP=5 LEFT=5 WIDTH=360
HEIGHT=180>
<H2>ARTICLE I</H2>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of speech, or of
the press; or the right of the people peaceably to assemble, and to petition the
government for a redress of grievances.
</P>
</LAYER>
</LAYER>
</BODY>
</HTML>
```

src

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

Setting the *layerObject*.src property of a layer that is a member of a layer family (that is, a family with at least one parent and one child) can be tricky business if you're not careful. Listing 31-7 presents a workspace for you to see how changing the src property of outer and inner layers affects the scenery.

When you first load the document, one outer layer contains one inner layer (each with a different background color). Control buttons on the page enable you to set the *layerObject*.src property of each layer independently. Changes to the inner layer content affect only that layer. Long content forces the inner layer to expand its depth, but the inner layer's view is automatically clipped by its parent layer.

Changing the outer layer content, however, removes the inner layer completely. Code in the following listing shows one way to examine for the presence of a particular layer before attempting to load new content in it. If the inner layer doesn't exist, the script creates a new layer on the fly to replace the original inner layer.

Listing 31-7: **Setting Nested Layer Source Content**

```
<HTML>
<HEAD>
<TITLE>Layer Source</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function loadOuter(doc) {
    document.outerDisplay.src = doc
}
function loadInner(doc) {
    var nested = document.outerDisplay.document.layers
    if (nested.length > 0) {
        // inner layer exists, so load content or restore
        if (doc) {
            nested[0].src = doc
        } else {
            restoreInner(nested[0])
        }
    } else {
        // prompt user about restoring inner layer
        if (confirm("The inner layer has been removed by loading an " +
        "outer document. Restore the original layers?")) {
            restoreLayers(doc)
        }
    }
}
function restoreLayers(doc) {
    // reset appearance of outer layer
    document.outerDisplay.bgColor = "coral"
    document.outerDisplay.resizeTo(370,190) // sets clip
    document.outerDisplay.document.write("")
    document.outerDisplay.document.close()
    // generate new inner layer
    var newInner = new Layer(360, document.layers["outerDisplay"])
    newInner.bgColor = "aquamarine"
    newInner.moveTo(5,5)
    if (doc) {
        // user clicked an inner content button
        newInner.src = doc
    } else {
        // return to pristine look
        restoreInner(newInner)
    }
    newInner.visibility = "show"
}
function restoreInner(inner) {
    inner.document.write("<HTML><BODY><P><B>Placeholder text for raw inner " +
    "layer.</B></P></BODY></HTML>")
    inner.document.close()
    inner.resizeTo(360,180) // sets clip
}
</SCRIPT>
```

```
</HEAD>
<BODY>
<B>Setting the <TT>layer.src</TT> Property of Nested Layers</B>
<HR>
Click the buttons to see what happens when you load new source documents into
the <FONT COLOR="coral">outer layer</FONT> and <FONT COLOR="aquamarine">inner
layer</FONT> objects.<P>
<LAYER TOP=100 BGCOLOR="coral">
<FORM>
Load into outer layer:<BR>
<INPUT TYPE="button" VALUE="Article I" onClick="loadOuter('article1.htm')"><BR>
<INPUT TYPE="button" VALUE="Entire Bill of Rights"
onClick="loadOuter('bofright.htm')"><BR>
</FORM>
</LAYER>
<LAYER TOP=220 BGCOLOR="aquamarine">
<FORM>
Load into inner layer:<BR>
<INPUT TYPE="button" VALUE="Article I" onClick="loadInner('article1.htm')"><BR>
<INPUT TYPE="button" VALUE="Entire Bill of Rights"
onClick="loadInner('bofright.htm')"><BR>
<INPUT TYPE="button" VALUE="Restore Original" onClick="loadInner()"><BR>
</FORM>
</LAYER>
<LAYER NAME="outerDisplay" BGCOLOR="coral" TOP=100 LEFT=200 WIDTH=370
HEIGHT=190>
    <LAYER NAME="innerDisplay" BGCOLOR="aquamarine" TOP=5 LEFT=5 WIDTH=360
HEIGHT=180>
    <P><B>Placeholder text for raw inner layer.</B></P>
    </LAYER>
</LAYER>
</BODY>
</HTML>
```

Restoring the original layers via script (as opposed to reloading the document) does not perform a perfect restoration. The key difference is that the scripts use the *layerObject*.resizeTo() method to set the layers to the height and width established by the <LAYER> tags that create the layers in the first place. This method, however, sets the clipping rectangle of the layer — not the layer's size. Therefore, if you use the script to restore the layers, loading the longer text file into either layer does not force the layer to expand to display all the content; the clipping region governs the view.

visibility

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

Use the page in Listing 31-8 to see how the *layerObject*.visibility property settings affect a pair of nested layers. When the page first loads, the default inherit setting is in effect. Changes you make to the outer layer by clicking the outer layer buttons affect the inner layer, but setting the inner layer's properties to hide or show severs the visibility relationship between parent and child. Listing 31-19 in the *JavaScript Bible* shows this example with IE5+ and NN6+ syntax.

Listing 31-8: Nested Layer Visibility Relationships

```
<HTML>
<HEAD>
<TITLE>Layer Source</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setOuterVis(type) {
    document.outerDisplay.visibility = type
}
function setInnerVis(type) {
    document.outerDisplay.document.innerDisplay.visibility = type
}
</SCRIPT>
</HEAD>
<BODY>
<B>Setting the <TT>layer.visibility</TT> Property of Nested Layers</B>
<HR>
Click the buttons to see what happens when you change the visibility of the
<FONT COLOR="coral">outer layer</FONT> and <FONT COLOR="aquamarine">inner
layer</FONT> objects.<P>
<LAYER TOP=100 BGCOLOR="coral">
<FORM>
Control outer layer property:<BR>
<INPUT TYPE="button" VALUE="Hide Outer Layer" onClick="setOuterVis('hide')"><BR>
<INPUT TYPE="button" VALUE="Show Outer Layer" onClick="setOuterVis('show')"><BR>
</FORM>
</LAYER>
<LAYER TOP=220 BGCOLOR="aquamarine">
<FORM>
Control inner layer property:<BR>
<INPUT TYPE="button" VALUE="Hide Inner Layer" onClick="setInnerVis('hide')"><BR>
<INPUT TYPE="button" VALUE="Show Inner Layer" onClick="setInnerVis('show')"><BR>
<INPUT TYPE="button" VALUE="Inherit Outer Layer"
onClick="setInnerVis('inherit')"><BR>
</FORM>
</LAYER>
<LAYER NAME="outerDisplay" BGCOLOR="coral" TOP=100 LEFT=200 WIDTH=370
HEIGHT=190>
    <LAYER NAME="innerDisplay" BGCOLOR="aquamarine" TOP=5 LEFT=5 WIDTH=360
HEIGHT=180>
    <P><B>Placeholder text for raw inner layer.</B></P>
    </LAYER>
</LAYER>
</BODY>
</HTML>
```

zIndex

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

The relationships among the three stacking property values can be difficult to visualize. Listing 31-9 offers a way to see the results of changing the *layerObject*.zIndex properties of three overlapping sibling layers. Figure 15-4 shows the beginning organization of layers after the page loads.

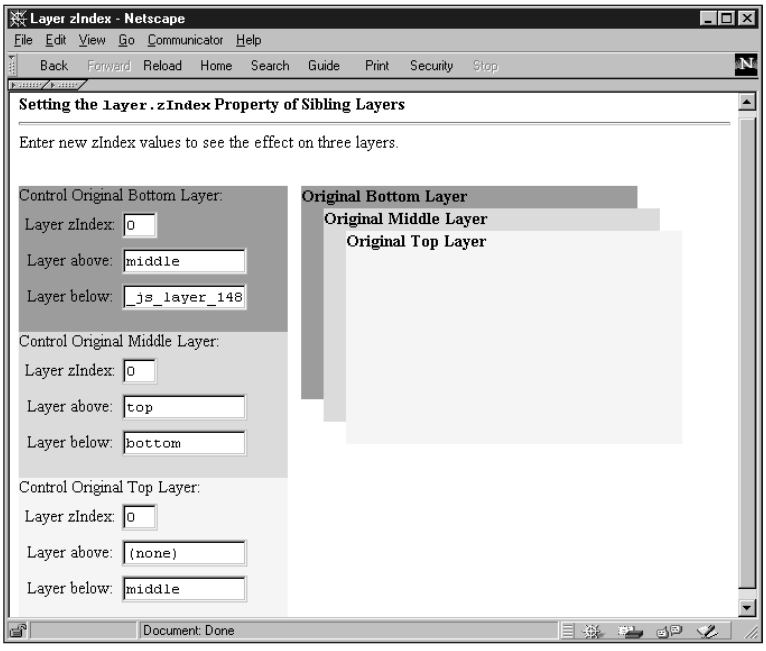


Figure 15-4: A place to play with zIndex property settings

The sequence of the <LAYER> tags in the document governs the original stacking order. Because the attribute is not set in the HTML, the initial values appear as zero for all three layers. But, as the page reveals, the *layerObject*.above and *layerObject*.below properties are automatically established. When a layer has no other layer object above it, the page shows (none). Also, if the layer below the bottom of the stack is the main window, a strange inner layer name is assigned (something like _js_layer_21).

To experiment with this page, first make sure you understand the *layerObject*.above and *layerObject*.below readings for the default order of the layers. Then, assign different orders to the layers with value sequences such as

3-2-1, 1-3-2, 2-2-2, and so on. Each time you enter one new value, check the actual layers to see if their stacking order changed and how that affected the other properties of all layers. Listing 31-20 in the *JavaScript Bible* shows how to achieve the same action with IE5+ and NN6+ syntax.

Listing 31-9: **Relationships Among zIndex, above, and below**

```
<HTML>
<HEAD>
<TITLE>Layer zIndex</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setZ(field) {
    switch (field.name) {
        case "top" :
            document.top.zIndex = parseInt(field.value)
            break
        case "mid" :
            document.middle.zIndex = parseInt(field.value)
            break
        case "bot" :
            document.bottom.zIndex = parseInt(field.value)
    }
    showValues()
}
function showValues() {
    document.layers[0].document.forms[0].bot.value = document.bottom.zIndex
    document.layers[1].document.forms[0].mid.value = document.middle.zIndex
    document.layers[2].document.forms[0].top.value = document.top.zIndex

    document.layers[0].document.forms[0].above.value = (document.bottom.above) ?
        document.bottom.above.name : "(none)"
    document.layers[1].document.forms[0].above.value = (document.middle.above) ?
        document.middle.above.name : "(none)"
    document.layers[2].document.forms[0].above.value = (document.top.above) ?
        document.top.above.name : "(none)"

    document.layers[0].document.forms[0].below.value = (document.bottom.below) ?
        document.bottom.below.name : "(none)"
    document.layers[1].document.forms[0].below.value = (document.middle.below) ?
        document.middle.below.name : "(none)"
    document.layers[2].document.forms[0].below.value = (document.top.below) ?
        document.top.below.name : "(none)"
}
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<B>Setting the <TT>layer.zIndex</TT> Property of Sibling Layers</B>
<HR>
Enter new zIndex values to see the effect on three layers.<P>
<LAYER TOP=90 WIDTH=240 BGCOLOR="coral">
```

```
<FORM>
Control Original Bottom Layer:<BR>
<TABLE>
<TR><TD ALIGN="right">Layer zIndex:</TD><TD><INPUT TYPE="text" NAME="bot" SIZE=3
onChange="setZ(this)"></TD></TR>
<TR><TD ALIGN="right">Layer above:</TD><TD><INPUT TYPE="text" NAME="above"
SIZE=13></TD></TR>
<TR><TD ALIGN="right">Layer below:</TD><TD><INPUT TYPE="text" NAME="below"
SIZE=13></TD></TR>
</TABLE>
</FORM>
</LAYER>
<LAYER TOP=220 WIDTH=240 BGCOLOR="aquamarine">
<FORM>
Control Original Middle Layer:<BR>
<TABLE>
<TR><TD ALIGN="right">Layer zIndex:</TD><TD><INPUT TYPE="text" NAME="mid" SIZE=3
onChange="setZ(this)"></TD></TR>
<TR><TD ALIGN="right">Layer above:</TD><TD><INPUT TYPE="text" NAME="above"
SIZE=13></TD></TR>
<TR><TD ALIGN="right">Layer below:</TD><TD><INPUT TYPE="text" NAME="below"
SIZE=13></TD></TR>
</TABLE></FORM>
</LAYER>
<LAYER TOP=350 WIDTH=240 BGCOLOR="yellow">
<FORM>
Control Original Top Layer:<BR>
<TABLE><TR><TD ALIGN="right">Layer zIndex:</TD><TD><INPUT TYPE="text" NAME="top"
SIZE=3 onChange="setZ(this)"></TD></TR>
<TR><TD ALIGN="right">Layer above:</TD><TD><INPUT TYPE="text" NAME="above"
SIZE=13></TD></TR>
<TR><TD ALIGN="right">Layer below:</TD><TD><INPUT TYPE="text" NAME="below"
SIZE=13></TD></TR>
</TABLE>
</FORM>
</LAYER>
<LAYER NAME="bottom" BGCOLOR="coral" TOP=90 LEFT=260 WIDTH=300 HEIGHT=190>
    <P><B>Original Bottom Layer</B></P>
</LAYER>
    <LAYER NAME="middle" BGCOLOR="aquamarine" TOP=110 LEFT=280 WIDTH=300
    HEIGHT=190>
    <P><B>Original Middle Layer</B></P>
</LAYER>
<LAYER NAME="top" BGCOLOR="yellow" TOP=130 LEFT=300 WIDTH=300 HEIGHT=190>
    <P><B>Original Top Layer</B></P>
</LAYER>
</LAYER>
</BODY>
</HTML>
```

Methods

load("*URL*", *newLayerWidth*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

Buttons in Listing 31-10 enable you to load short and long documents into a layer. The first two buttons don't change the width (in fact, the second parameter to *layerObject*.load() is the *layerObject*.clip.left value). For the second two buttons, a narrower width than the original is specified. Click the Restore button frequently to return to a known state.

Listing 31-10: **Loading Documents into Layers**

```
<HTML>
<HEAD>
<TITLE>Layer Loading</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function loadDoc(URL,width) {
    if (!width) {
        width = document.myLayer.clip.width
    }
    document.myLayer.load(URL, width)
}
</SCRIPT>
</HEAD>
<BODY>
<B>Loading New Documents</B>
<HR>
<LAYER TOP=90 WIDTH=240 BGCOLOR="yellow">
<FORM>
Loading new documents:<BR>
<INPUT TYPE="button" VALUE="Small Doc/Existing Width"
onClick="loadDoc('article1.htm')"><BR>
<INPUT TYPE="button" VALUE="Large Doc/Existing Width"
onClick="loadDoc('bofright.htm')"><P>
<INPUT TYPE="button" VALUE="Small Doc/Narrower Width"
onClick="loadDoc('article1.htm',200)"><BR>
<INPUT TYPE="button" VALUE="Large Doc/Narrower Width"
onClick="loadDoc('bofright.htm',200)"><P>
<INPUT TYPE="button" VALUE="Restore" onClick="location.reload()"></FORM>
</LAYER>
<LAYER NAME="myLayer" BGCOLOR="yellow" TOP=90 LEFT=300 WIDTH=300 HEIGHT=190>
    <P><B>Text loaded in original document.</B></P>
```

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

moveAbove(*layerObject*)
moveBelow(*layerObject*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

You can see the *layerObject*.moveAbove() method at work in Listing 31-1.

moveBy(*deltaX*,*deltaY*)
moveTo(*x*,*y*)
moveToAbsolute(*x*,*y*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

Listing 31-11 shows a demonstration of the *layerObject*.moveTo() method. It is a simple script that enables you to click and drag a layer around the screen. The script employs the coordinate values of the mouseMove event; after compensating for the offset within the layer at which the click occurs, the script moves the layer to track the mouse action.

I want to present this example for an additional reason: to explain an important user interface difference between Windows and Macintosh versions of NN4. In Windows versions, you can click and hold the mouse button down on an object and let the object receive all the mouseMove events as you drag the cursor around the screen. On the Macintosh, however, NN4 tries to compensate for the lack of a second mouse button by popping up a context-sensitive menu at the cursor position when the user holds the mouse button down for more than just a click. To prevent the pop-up menu from appearing, the engage() method invoked by the onMouseDown event handler ends with return false.

Notice in the following listing how the layer captures a number of mouse events. Each one plays an important role in creating a mode that is essentially like a mouseStillDown event (which doesn't exist in NN4's event model). The mouseDown event sets a Boolean flag (engaged) indicating that the user clicked down in the

layer. At the same time, the script records how far away from the layer's top-left corner the `mouseDown` event occurred. This offset information is needed so that any setting of the layer's location takes this offset into account (otherwise, the top-left corner of the layer would jump to the cursor position and be dragged from there).

During the drag (`mouseDown` events firing with each mouse movement), the `dragIt()` function checks whether the drag mode is engaged. If so, the layer is moved to the page location calculated by subtracting the original downstroke offset from the `mouseMove` event location on the page. When the user releases the mouse button, the `mouseUp` event turns off the drag mode Boolean value. Listing 31-21 in the *JavaScript Bible* shows a version of this example for IE5+ and NN6.

Listing 31-11: **Dragging a Layer**

```
<HTML>
<HEAD>
<TITLE>Layer Dragging</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var engaged = false
var offsetX = 0
var offsetY = 0
function dragIt(e) {
    if (engaged) {
        document.myLayer.moveTo(e.pageX - offsetX, e.pageY - offsetY)
    }
}
function engage(e) {
    engaged = true
    offsetX = e.pageX - document.myLayer.left
    offsetY = e.pageY - document.myLayer.top
    return false
}
function release() {
    engaged = false
}
</SCRIPT>
</HEAD>
<BODY>
<B>Dragging a Layer</B>
<HR>
<LAYER NAME="myLayer" BGCOLOR="lightgreen" TOP=90 LEFT=100 WIDTH=300 HEIGHT=190>
    <P><B>Drag me around the window.</B></P>
</LAYER>
<SCRIPT LANGUAGE="JavaScript">
document.myLayer.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP |
Event.MOUSEMOVE)
document.myLayer.onMouseDown = engage
document.myLayer.onMouseUp = release
document.myLayer.onMouseMove = dragIt
</SCRIPT>
</BODY>
</HTML>
```

resizeBy(*deltaX*,*deltaY*)
resizeTo(*width*,*height*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Example

It is important to understand the ramifications of content flow when these two methods resize a layer. Listing 31-12a (and the companion document Listing 31-12b) shows you how to set the lower-right corner of a layer to be dragged by a user for resizing the layer (much like grabbing the resize corner of a document window). Three radio buttons enable you to choose whether and when the content should be redrawn to the layer — never, after resizing, or during resizing.

Event capture is very much like that in Listing 31-11 for layer dragging. The primary difference is that drag mode is engaged only when the mouse event takes place in the region of the lower-right corner. A different kind of offset value is saved here because, for resizing, the script needs to know the mouse event offset from the right and bottom edges of the layer.

Condition statements in the `resizeIt()` and `release()` functions verify whether a specific radio button is checked to determine when (or if) the content should be redrawn. I designed this page with the knowledge that its content might be redrawn. Therefore, I built the content of the layer as a separate HTML document that loads in the `<LAYER>` tag.

Redrawing the content requires reloading the document into the layer. I use the *`layerObject`*`.load()` method because I want to send the current *`layerObject`*`.clip.width` as a parameter for the width of the clip region to accommodate the content as it loads.

An important point to know about reloading content into a layer is that all property settings for the layer's event capture are erased when the document loads. Overcoming this behavior requires setting the layer's `onLoad` event handler to set the layer's event capture mechanism. If the layer event capturing is specified as part of the statements at the end of the document, the layer ignores some important events needed for the dynamic resizing after the document reloads the first time.

As you experiment with the different ways to resize and redraw, you see that redrawing during resizing is a slow process because of the repetitive loading (from cache) needed each time. On slower client machines, it is easy for the cursor to outrun the layer region, causing the layer to not get `mouseOver` events at all. It may not be the best-looking solution, but I prefer to redraw after resizing the layer.

Listing 31-22 in the *JavaScript Bible* shows a version designed for the IE5+ and NN6 object models. Because content automatically reflows in those browsers, you do not have to load the content of the positioned element from an external document.

Listing 31-12a: **Resizing a Layer**

```
<HTML>
<HEAD>
<TITLE>Layer Resizing</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var engaged = false
var offsetX = 0
var offsetY = 0
function resizeIt(e) {
    if (engaged) {
        document.myLayer.resizeTo(e.pageX + offsetX, e.pageY + offsetY)
        if (document.forms[0].redraw[2].checked) {
            document.myLayer.load("lst31-12b.htm", document.myLayer.clip.width)
        }
    }
}
function engage(e) {
    if (e.pageX > (document.myLayer.clip.right - 10) &&
        e.pageY > (document.myLayer.clip.bottom - 10)) {
        engaged = true
        offsetX = document.myLayer.clip.right - e.pageX
        offsetY = document.myLayer.clip.bottom - e.pageY
    }
}
function release() {
    if (engaged && document.forms[0].redraw[1].checked) {
        document.myLayer.load("lst31-12b.htm", document.myLayer.clip.width)
    }
    engaged = false
}
function grabEvents() {
    document.myLayer.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP |
Event.MOUSEMOVE)
}
</SCRIPT>
</HEAD>
<BODY>
<B>Resizing a Layer</B>
<HR>
<FORM>
Redraw layer content:<BR>
<INPUT TYPE="radio" NAME="redraw" CHECKED>Never
<INPUT TYPE="radio" NAME="redraw">After resize
<INPUT TYPE="radio" NAME="redraw">During resize
</FORM>
<LAYER NAME="myLayer" SRC="lst31-12b.htm" BGCOLOR="lightblue" TOP=120 LEFT=100
WIDTH=300 HEIGHT=190 onLoad="grabEvents()">
</LAYER>
<SCRIPT LANGUAGE="JavaScript">
```

```
document.myLayer.onMouseDown = engage
document.myLayer.onMouseUp = release
document.myLayer.onMouseMove = resizeIt
</SCRIPT>
</BODY>
</HTML>
```

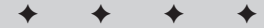
Listing 31-12b: **Content for the Resizable Layer**

```
<HTML>
<BODY>
    <P><B>Resize me by dragging the lower-right corner.</B></P>
    <SCRIPT LANGUAGE="JavaScript">
    if (navigator.userAgent.indexOf("Mac") != -1) {
        document.write("(Mac users: Ctrl-Click me first; then Click to stop
dragging.)")
    }
    </SCRIPT>
</BODY>
</HTML>
```

✦ ✦ ✦

CHAPTER 16

String and Number Objects (Chapters 34 and 35)



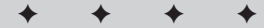
In This Chapter

Parsing text at the character level

Performing search-and-replace operations with regular expressions

Converting between character codes and text

Setting number format and precision



Knowing how to manipulate strings of text characters is a vital programming skill. You may not have to do it all the time, but you should be fully aware of the possibilities for this manipulation that are built into whatever programming language you use. In JavaScript (as in any object-based or object-oriented language), strings are objects that have numerous properties and methods to assist in assembling, tearing apart, extracting, and copying chunks of strings.

Any characters that users enter into text boxes become parts of string objects. In IE4+ and NN6, text inside HTML element tags can be treated as strings. In IE4+, you can even work with the HTML tags as strings. Therefore, of all the core language objects to implant in your scripting consciousness, the string object is it (arrays, whose examples come in the next chapter of this book, rank Number Two on the list).

Numbers are much less frequently thought of as objects because they tend to be used as-is for calculations. JavaScript 1.5 in recent browsers, however, endows the number object with practical methods, especially one that (finally) offers built-in control over the number of digits displayed to the right of the decimal point for floating-point numbers.

When examples in this chapter encourage you to enter a sequence of expressions in The Evaluator, be sure to follow through with every step. But also make sure you understand the results of each expression in order to visualize the particular method operates.

Examples Highlights

- ✦ Study the code and operation of Listing 34-2 to see how to use JavaScript to convert characters to character codes and vice versa. Converting ASCII or Unicode numeric values to their corresponding characters requires the `String.fromCharCode()` method of the static `String` object.
- ✦ Compare the sequence of steps for the *`string`*`.indexOf()` and *`string`*`.lastIndexOf()` methods to grasp fully the behavior of each and the differences between them.
- ✦ Listing 34-4 lets you experiment with the *`string`*`.replace()` and *`string`*`.search()` methods, both of which utilize regular expression powers available in JavaScript 1.2 of NN4+ and IE4+. Notice how the script functions assemble the regular expression objects with global modifiers.
- ✦ Walk through the steps of the *`string`*`.split()` method example to convert a string to an array.
- ✦ Compare the behaviors and coding of Listings 34-6 and 34-7 to distinguish the subtle differences between the *`string`*`.substr()` and *`string`*`.substring()` methods.
- ✦ Study the example for *`string`*`.toLowerCase()` and *`string`*`.toUpperCase()` to see how to remove case sensitivity issues for some operations.
- ✦ Convert a long floating-point number to a dollars-and-cents string by following the steps for the *`number`*`.toFixed()` method.

String Object

Properties

constructor

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to test the value of the `constructor` property. Enter the following statements into the top text box:

```
a = new String("abcd")
a.constructor == String
a.constructor == Number
```

Parsing methods

string.charAt(*index*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Enter each of the following statements into the top text box of The Evaluator:

```
a = "banana daiquiri"
a.charAt(0)
a.charAt(5)
a.charAt(6)
a.charAt(20)
```

Results from each of the `charAt()` methods should be `b`, `a` (the third "a" in "banana"), a space character, and an empty string, respectively.

string.charCodeAt([*index*])
String.fromCharCode(*num1* [, *num2* [, ... *numn*]])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Listing 34-2 provides examples of both methods on one page. Moreover, because one of the demonstrations relies on the automatic capture of selected text on the page, the scripts include code to accommodate the different handling of selection events and capture of the selected text in Navigator and Internet Explorer 4.

After you load the page, select part of the body text anywhere on the page. If you start the selection with the lowercase letter "a," the character code displays as 97. If you select no text, the result is `NaN`.

Try entering numeric values in the three fields at the bottom of the page. Values below 32 are ASCII control characters that most fonts represent as hollow squares. But try all other values to see what you get. Notice that the script passes all three values as a group to the `String.fromCharCode()` method, and the result is a combined string. Thus, Figure 16-1 shows what happens when you enter the uppercase ASCII values for a three-letter animal name.

Listing 34-2: **Character Conversions**

```
<HTML>
<HEAD>
<TITLE>Character Codes</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var isNav = (navigator.appName == "Netscape")
var isNav4 = (isNav && parseInt(navigator.appVersion == 4))
function showCharCode() {
    if (isNav) {
        var theText = document.getSelection()
    } else {
        var theText = document.selection.createRange().text
    }
    if (theText) {
        document.forms[0].charCodeDisplay.value = theText.charCodeAt()
    } else {
        document.forms[0].charCodeDisplay.value = " "
    }
}
function showString(form) {
    form.result.value =
String.fromCharCode(form.entry1.value,form.entry2.value,form.entry3.value)
}
if (isNav4) {
    document.captureEvents(Event.MOUSEUP)
}
document.onmouseup = showCharCode
</SCRIPT>
</HEAD>
<BODY>
<B>Capturing Character Codes</B>
<FORM>
Select any of this text, and see the character code of the first character.<P>
Character Code:<INPUT TYPE="text" NAME="charCodeDisplay" SIZE=3><BR>
<HR>
<B>Converting Codes to Characters</B><BR>
Enter a value  0-255:<INPUT TYPE="text" NAME="entry1" SIZE=4><BR>
Enter a value  0-255:<INPUT TYPE="text" NAME="entry2" SIZE=4><BR>
Enter a value  0-255:<INPUT TYPE="text" NAME="entry3" SIZE=4><BR>
<INPUT TYPE="button" VALUE="Show String" onClick="showString(this.form)">
Result:<INPUT TYPE="text" NAME="result" SIZE=5>
</FORM>
</BODY>
</HTML>
```

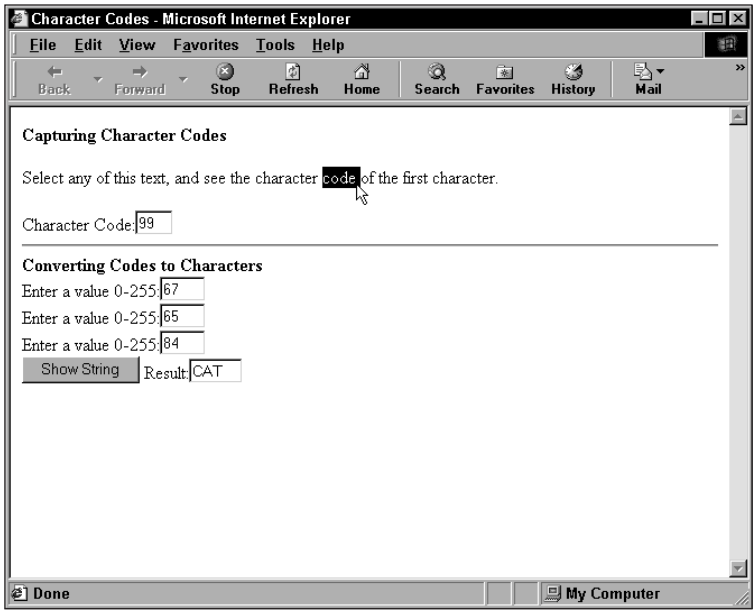


Figure 16-1: Conversions from text characters to ASCII values and vice versa

string.indexOf(*searchString* [, *startIndex*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Enter each of the following statements (up to, but not including the "//" comment symbols) into the top text box of The Evaluator (you can simply replace the parameters of the `indexOf()` method for each statement after the first one). Compare your results with the results shown below.

```
a = "bananas"
a.indexOf("b")      // result = 0 (index of 1st letter is zero)
a.indexOf("a")      // result = 1
a.indexOf("a",1)    // result = 1 (start from 2nd letter)
a.indexOf("a",2)    // result = 3 (start from 3rd letter)
a.indexOf("a",4)    // result = 5 (start from 5th letter)
a.indexOf("nan")    // result = 2
a.indexOf("nas")    // result = 4
a.indexOf("s")      // result = 6
a.indexOf("z")      // result = -1 (no "z" in string)
```

string.lastIndexOf(*searchString*[, *startIndex*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Enter each of the following statements (up to, but not including the "//" comment symbols) into the top text box of The Evaluator (you can simply replace the parameters of the `lastIndexOf()` method for each statement after the first one). Compare your results with the results shown below.

```
a = "bananas"
a.lastIndexOf("b")    // result = 0 (index of 1st letter is zero)
a.lastIndexOf("a")    // result = 5
a.lastIndexOf("a",1)  // result = 1 (from 2nd letter toward the front)
a.lastIndexOf("a",2)  // result = 1 (start from 3rd letter working toward front)
a.lastIndexOf("a",4)  // result = 3 (start from 5th letter)
a.lastIndexOf("nan")  // result = 2 [except for -1 Nav 2.0 bug]
a.lastIndexOf("nas")  // result = 4
a.lastIndexOf("s")    // result = 6
a.lastIndexOf("z")    // result = -1 (no "z" in string)
```

string.match(*regExpression*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

To help you understand the *string*.match() method, Listing 34-3 provides a workshop area for experimentation. Two fields occur for data entry: the first is for the long string to be examined by the method; the second is for a regular expression. Some default values are provided in case you're not yet familiar with the syntax of regular expressions (see Chapter 38 of the *JavaScript Bible*). A check box lets you specify whether the search through the string for matches should be case-sensitive. After you click the "Execute match()" button, the script creates a regular expression object out of your input, performs the *string*.match() method on the big string, and reports two kinds of results to the page. The primary result is a string version of the array returned by the method; the other is a count of items returned.

Listing 34-3: **Regular Expression Match Workshop**

```
<HTML>
<HEAD>
<TITLE>Regular Expression Match</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function doMatch(form) {
    var str = form.entry.value
    var delim = (form.caseSens.checked) ? "/g" : "/gi"
    var regexp = eval("/" + form.regexp.value + delim)
    var resultArray = str.match(regexp)
    if (resultArray) {
        form.result.value = resultArray.toString()
        form.count.value = resultArray.length
    } else {
        form.result.value = "<no matches>"
        form.count.value = ""
    }
}
</SCRIPT>
</HEAD>
<BODY>
<B>String Match with Regular Expressions</B>
<HR>
<FORM>
Enter a main string:<INPUT TYPE="text" NAME="entry" SIZE=60
  VALUE="Many a maN and womAN have meant to visit GerMAny."><BR>
Enter a regular expression to match:<INPUT TYPE="text" NAME="regexp" SIZE=25
  VALUE="\wa\w">
<INPUT TYPE="checkbox" NAME="caseSens">Case-sensitive<P>
<INPUT TYPE="button" VALUE="Execute match()" onClick="doMatch(this.form)">
<INPUT TYPE="reset"><P>
Result:<INPUT TYPE="text" NAME="result" SIZE=40><BR>
Count:<INPUT TYPE="text" NAME="count" SIZE=3><BR>
</FORM>
</BODY>
</HTML>
```

The default value for the main string has unusual capitalization intentionally. The capitalization lets you see more clearly where some of the matches come from. For example, the default regular expression looks for any three-character string that has the letter "a" in the middle. Six string segments match that expression. With the help of capitalization, you can see where each of the four strings containing "man" are extracted from the main string. The following table lists some other regular expressions to try with the default main string.

RegExp	*Description*
`man`	Both case-sensitive and not
`man\b`	Where "man" is at the end of a word
`\bman`	Where "man" is at the start of a word
`me*an`	Where zero or more "e" letters occur between "m" and "a"
`.a.`	Where "a" is surrounded by any one character (including space)
`\sa\s`	Where "a" is surrounded by a space on both sides
`z`	Where a "z" occurs (none in the default string)

In the scripts for Listing 34-3, if the `string.match()` method returns `null`, you are informed politely, and the count field is emptied.

string.replace(*regExpression, replaceString*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

The page in Listing 34-4 lets you practice with the *`string`*`.replace()` and *`string`*`.search()` methods and regular expressions in a friendly environment. The source text is a five-line excerpt from *Hamlet*. You can enter the regular expression to search for, and the replacement text as well. Note that the script completes the job of creating the regular expression object, so that you can focus on the other special characters used to define the matching string. All replacement activities act globally, because the `g` parameter is automatically appended to any expression you enter.

Default values in the fields replace the contraction 'tis with "it is" after you click the "Execute replace( )" button (see Figure 16-2). Notice that the backslash character in front of the apostrophe of 'tis (in the string assembled in `mainString`) makes the apostophe a non-word boundary, and thus allows the `\B't` regular expression to find a match there. As described in the section on the *`string`*`.search()` method, the button connected to that method returns the offset character number of the matching string (or `-1` if no match occurs).

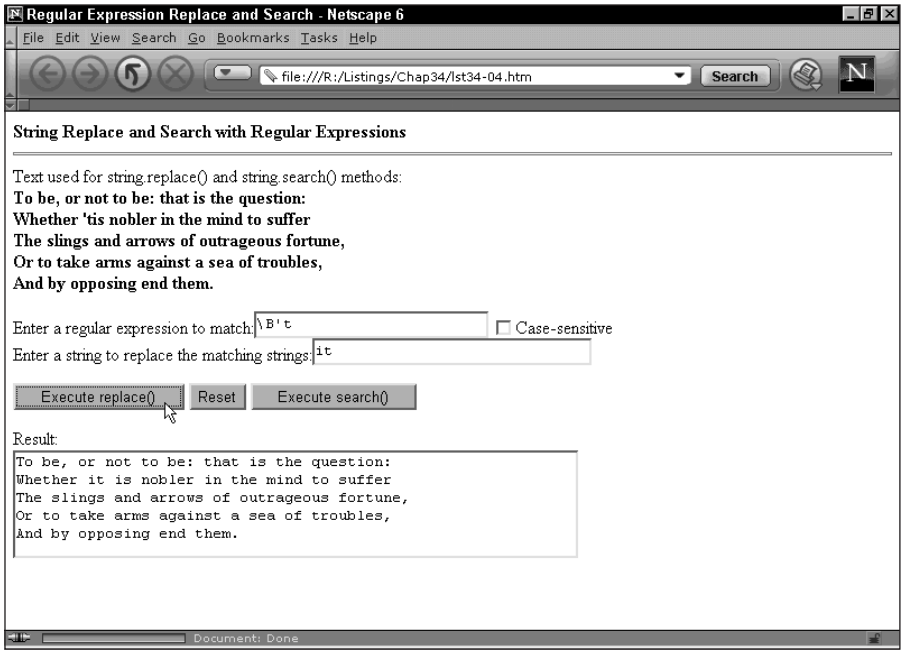


Figure 16-2: Using the default replacement regular expression

You could modify the listing so that it actually replaces text in the HTML paragraph for IE4+ and NN6. The steps include wrapping the paragraph in its own element (for example, a SPAN), and invoking the `replace()` method on the `innerHTML` of that element. Assign the results to the `innerHTML` property of that element to complete the job.

Listing 34-4: **Lab for string.replace() and string.search()**

```
<HTML>
<HEAD>
<TITLE>Regular Expression Replace and Search</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var mainString = "To be, or not to be: that is the question:\n"
mainString += "Whether \'tis nobler in the mind to suffer\n"
mainString += "The slings and arrows of outrageous fortune,\n"
mainString += "Or to take arms against a sea of troubles,\n"
mainString += "And by opposing end them."

function doReplace(form) {
    var replaceStr = form.replaceEntry.value
    var delim = (form.caseSens.checked) ? "/g" : "/gi"
    var regexp = eval("/" + form.regexp.value + delim)
    form.result.value = mainString.replace(regexp, replaceStr)
}
```

Continued

Listing 34-4 *(continued)*

```
function doSearch(form) {
    var replaceStr = form.replaceEntry.value
    var delim = (form.caseSens.checked) ? "/g" : "/gi"
    var regexp = eval("/" + form.regexp.value + delim)
    form.result.value = mainString.search(regexp)
}
</SCRIPT>
</HEAD>
<BODY>
<B>String Replace and Search with Regular Expressions</B>
<HR>
Text used for string.replace() and string.search() methods:<BR>
<B>To be, or not to be: that is the question:<BR>
Whether 'tis nobler in the mind to suffer<BR>
The slings and arrows of outrageous fortune,<BR>
Or to take arms against a sea of troubles,<BR>
And by opposing end them.</B>

<FORM>
Enter a regular expression to match:<INPUT TYPE="text" NAME="regexp" SIZE=25
VALUE="\B't">
<INPUT TYPE="checkbox" NAME="caseSens">Case-sensitive<BR>
Enter a string to replace the matching strings:<INPUT TYPE="text"
NAME="replaceEntry" SIZE=30 VALUE="it "><P>
<INPUT TYPE="button" VALUE="Execute replace()" onClick="doReplace(this.form)">
<INPUT TYPE="reset">
<INPUT TYPE="button" VALUE="Execute search()" onClick="doSearch(this.form)"><P>
Result:<BR>
<TEXTAREA NAME="result" COLS=60 ROWS=5 WRAP="virtual"></TEXTAREA>
</FORM>
</BODY>
</HTML>
```

string.search(*regExpression*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Listing 34-4, for the *string*.replace() method, also provides a laboratory to experiment with the *string*.search() method.

string.slice(*startIndex* [, *endIndex*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

With Listing 34-5, you can try several combinations of parameters with the *string*.slice() method (see Figure 16-3). A base string is provided (along with character measurements). Select from the different choices available for parameters and study the outcome of the slice.

Listing 34-5: **Slicing a String**

```
<HTML>
<HEAD>
<TITLE>String Slicing and Dicing, Part I</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var mainString = "Electroencephalograph"
function showResults() {
    var form = document.forms[0]
    var param1 = parseInt(form.param1.options[form.param1.selectedIndex].value)
    var param2 = parseInt(form.param2.options[form.param2.selectedIndex].value)
    if (!param2) {
        form.result1.value = mainString.slice(param1)
    } else {
        form.result1.value = mainString.slice(param1, param2)
    }
}
</SCRIPT>
</HEAD>
<BODY onLoad="showResults()">
<B>String slice() Method</B>
<HR>
Text used for the methods:<BR>
<FONT SIZE=+1><TT><B>Electroencephalograph<BR>
----5----5----5----5-</B></TT></FONT>
<TABLE BORDER=1>
<FORM>
<TR><TH>String Method</TH><TH>Method Parameters</TH><TH>Results</TH></TR>
<TR>
<TD>string.slice()</TD><TD ROWSPAN=3 VALIGN=middle>
( <SELECT NAME="param1" onChange="showResults()">
    <OPTION VALUE=0>0
    <OPTION VALUE=1>1
    <OPTION VALUE=2>2
    <OPTION VALUE=3>3
    <OPTION VALUE=5>5
```

Continued

Listing 34-5 *(continued)*

```
</SELECT>,
<SELECT NAME="param2" onChange="showResults()">
    <OPTION >(None)
    <OPTION VALUE=5>5
    <OPTION VALUE=10>10
    <OPTION VALUE=-1>-1
    <OPTION VALUE=-5>-5
    <OPTION VALUE=-10>-10
</SELECT> ) </TD>
<TD><INPUT TYPE="text" NAME="result1" SIZE=25></TD>
</TR>
</FORM>
</TABLE>
</BODY>
</HTML>
```

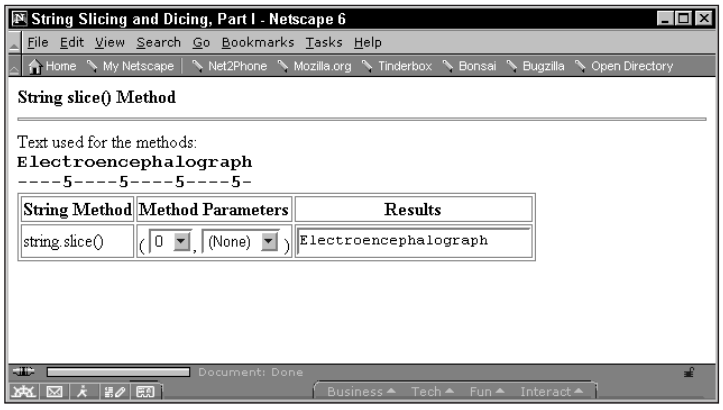


Figure 16-3: Lab for exploring the string.slice() method

string.split("*delimiterCharacter*" [, *limitInteger*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓			✓	✓	✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to see how the `string.split()` method works. Begin by assigning a comma-delimited string to a variable:

```
a = "Anderson,Smith,Johnson,Washington"
```

Now split the string at comma positions so that the string pieces become items in an array, saved as `b`:

```
b = a.split(",")
```

To prove that the array contains four items, inspect the array's `length` property:

```
b.length    // result: 4
```

string.substr(*start* [, *length*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Listing 34-6 lets you experiment with a variety of values to see how the *string*.substr() method works.

Listing 34-6: **Reading a Portion of a String**

```
<HTML>
<HEAD>
<TITLE>String Slicing and Dicing, Part II</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var mainString = "Electroencephalograph"
function showResults() {
    var form = document.forms[0]
    var param1 = parseInt(form.param1.options[form.param1.selectedIndex].value)
    var param2 = parseInt(form.param2.options[form.param2.selectedIndex].value)
    if (!param2) {
        form.result1.value = mainString.substr(param1)
    } else {
        form.result1.value = mainString.substr(param1, param2)
    }
}
</SCRIPT>
</HEAD>
<BODY onLoad="showResults()">
<B>String substr() Method</B>
<HR>
Text used for the methods:<BR>
<FONT SIZE=+1><TT><B>Electroencephalograph<BR>
----5----5----5----5-</B></TT></FONT>
<TABLE BORDER=1>
<FORM>
<TR><TH>String Method</TH><TH>Method Parameters</TH><TH>Results</TH></TR>
<TR>
```

Continued

Listing 34-6 *(continued)*

```
<TD>string.substr()</TD><TD ROWSPAN=3 VALIGN=middle>
( <SELECT NAME="param1" onChange="showResults()">
    <OPTION VALUE=0>0
    <OPTION VALUE=1>1
    <OPTION VALUE=2>2
    <OPTION VALUE=3>3
    <OPTION VALUE=5>5
</SELECT>,
<SELECT NAME="param2" onChange="showResults()">
    <OPTION >(None)
    <OPTION VALUE=5>5
    <OPTION VALUE=10>10
    <OPTION VALUE=20>20
</SELECT> ) </TD>
<TD><INPUT TYPE="text" NAME="result1" SIZE=25></TD>
</TR>
</FORM>
</TABLE>
</BODY>
</HTML>
```

string.substring(*indexA, indexB*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

Listing 34-7 lets you experiment with a variety of values to see how the *string*.substring() method works. If you are using Navigator 4, try changing the LANGUAGE attribute of the script to JavaScript1.2 and see the different behavior when you set the parameters to 5 and 3. The parameters switch themselves, essentially letting the second index value become the beginning of the extracted substring.

Listing 34-7: **Reading a Portion of a String**

```
<HTML>
<HEAD>
<TITLE>String Slicing and Dicing, Part III</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var mainString = "Electroencephalograph"
function showResults() {
```

```
    var form = document.forms[0]
    var param1 = parseInt(form.param1.options[form.param1.selectedIndex].value)
    var param2 = parseInt(form.param2.options[form.param2.selectedIndex].value)
    if (!param2) {
        form.result1.value = mainString.substring(param1)
    } else {
        form.result1.value = mainString.substring(param1, param2)
    }
}
</SCRIPT>
</HEAD>
<BODY onLoad="showResults()">
<B>String substr() Method</B>
<HR>
Text used for the methods:<BR>
<FONT SIZE=+1><TT><B>Electroencephalograph<BR>
----5----5----5----5-</B></TT></FONT>
<TABLE BORDER=1>
<FORM>
<TR><TH>String Method</TH><TH>Method Parameters</TH><TH>Results</TH></TR>
<TR>
<TD>string.substring()</TD><TD>
( <SELECT NAME="param1" onChange="showResults()">
    <OPTION VALUE=0>0
    <OPTION VALUE=1>1
    <OPTION VALUE=2>2
    <OPTION VALUE=3>3
    <OPTION VALUE=5>5
</SELECT>,
<SELECT NAME="param2" onChange="showResults()">
    <OPTION >(None)
    <OPTION VALUE=3>3
    <OPTION VALUE=5>5
    <OPTION VALUE=10>10
</SELECT> ) </TD>
<TD><INPUT TYPE="text" NAME="result1" SIZE=25></TD>
</TR>
</FORM>
</TABLE>
</BODY>
</HTML>
```

string.toLowerCase()
string.toUpperCase()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility	✓	✓	✓	✓	✓	✓	✓	✓	✓

Example

You can use the `toLowerCase()` and `toUpperCase()` methods on literal strings, as follows:

```
var newString = "HTTP://www.Netscape.COM".toLowerCase()
    // result = "http://www.netscape.com"
```

The methods are also helpful in comparing strings when case is not important, as follows:

```
if (guess.toUpperCase() == answer.toUpperCase()) {...}
    // comparing strings without case sensitivity
```

string.toString()
string.valueOf()

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Examples

Use The Evaluator to test the `valueOf()` method. Enter the following statements into the top text box and examine the values that appear in the Results field:

```
a = new String("hello")
typeof a
b = a.valueOf()
typeof b
```

Because all other JavaScript core objects also have the `valueOf()` method, you can build generic functions that receive a variety of object types as parameters, and the script can branch its code based on the type of value that is stored in the object.

Number Object

Properties

MAX_VALUE
MIN_VALUE
NEGATIVE_INFINITY
POSITIVE_INFINITY

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓		✓	✓	✓	✓

Example

Enter each of the four `Number` object expressions into the top text field of The Evaluator to see how the browser reports each value.

```
Number.MAX_VALUE
Number.MIN_VALUE
Number.NEGATIVE_INFINITY
Number.POSITIVE_INFINITY
```

Methods

number.toExponential(*fractionDigits*)
number.toFixed(*fractionDigits*)
number.toPrecision(*precisionDigits*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility				✓					✓

Example

You can use The Evaluator to experiment with all three of these methods with a variety of parameter values. Before invoking any method, be sure to assign a numeric value to one of the built-in global variables in The Evaluator (`a` through `z`).

```
a = 10/3
a.toFixed(4)
"$" + a.toFixed(2)
```

None of these methods works with number literals (for example, `123.toExponential(2)` does not work).

number.toString([*radix*])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓		✓	✓	✓	✓

Example

Use The Evaluator to experiment with the `toString()` method. Assign the number `12` to the variable `a` and see how the number is converted to strings in a variety of number bases:

```
a = 12
a.toString()    // base 10
a.toString(2)
a.toString(16)
```

✦ ✦ ✦

CHAPTER 17

The Array Object (Chapter 37)

In This Chapter

Converting an array to a delimited string

Sorting arrays

Combining arrays and replacing items in an array

Whenever you are faced with having to manage any kind of list or series of related data chunks, the first technique to turn to is stuffing those chunks into an array. Once the data is inside an array, your scripts can then perform quick and easy lookups, based on `for` loops through numerically indexed arrays, or via instant searching with the help of string indexes (à la Java hash tables).

As the examples in this chapter demonstrate, the JavaScript array object features numerous methods to facilitate managing the data inside an array. It also helps that JavaScript is loose enough to allow arrays to grow or shrink as their data requires.

Perhaps the two most important features of JavaScript arrays to have in your hip pocket are converting arrays to delimited string objects and sorting. Conversion to strings is important when you wish to transport data from an array to another venue that passes only strings, such as passing data to another page via the URL search string. At the receiving end, a script converts the search string to an array through the inverse operation provided by the *`string`*`.split()` method.

JavaScript's array sorting feature is remarkably powerful and flexible. Even if the array consists of objects, you can sort the array based on values assigned to properties of those objects.

Examples Highlights

- Convert an array into a delimited string via the code shown in Listing 37-7.
- To flip the order of an array without resorting to sorting, see Example 37-8 for the *`array`*`.reverse()` method.
- Listing 37-9 demonstrates a few important aspects of the *`array`*`.sort()` method. In addition to the traditional alphabetical sorting, one of the sorting functions operates on the `length` property of the string object stored in each entry of the array. Powerful stuff with very little code.

✦ Walk through the steps for the *array*.splice() method to observe how JavaScript in NN4+ and IE5.5+ can replace entries inside an array. One example replaces three items with one, indicating that you are not bound to maintaining the same array length.

Array Object Methods

array.concat(*array2*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓			✓	✓	✓

Example

Listing 37-6 is a bit complex, but it demonstrates both how arrays can be joined with the *array*.concat() method and how values and objects in the source arrays do or do not propagate based on their data type. The page is shown in Figure 17-1.

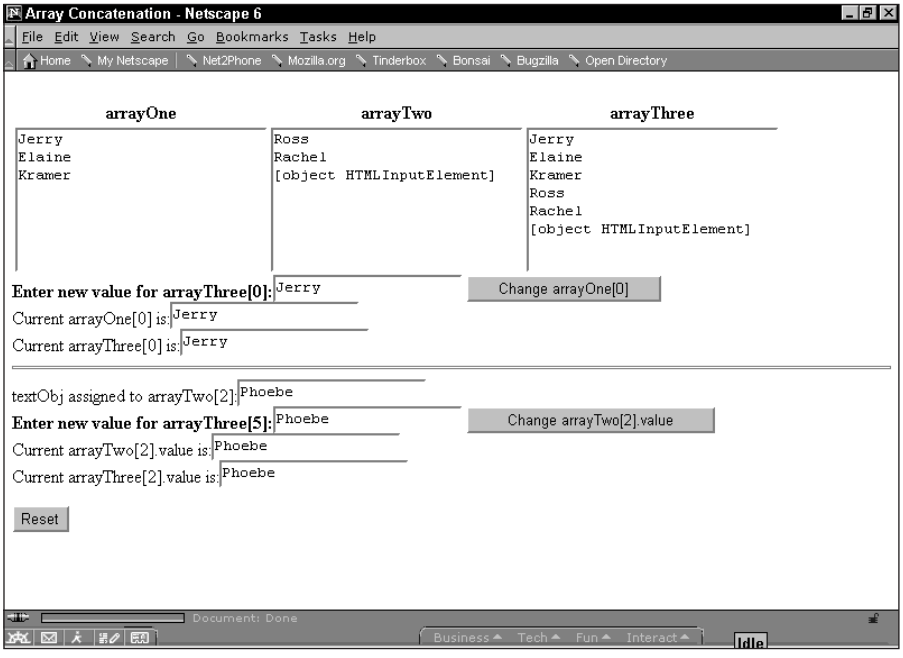


Figure 17-1: Object references remain "alive" in a concatenated array.

After you load the page, you see readouts of three arrays. The first array consists of all string values; the second array has two string values and a reference to a form

object on the page (a textbox named "original" in the HTML). In the initialization routine of this page, not only are the two source arrays created, but they are joined with the *array*.concat() method, and the result is shown in the third box. To show the contents of these arrays in columns, I use the *array*.join() method, which brings the elements of an array together as a string delimited in this case by a return character — giving us an instant column of data.

Two series of fields and buttons let you experiment with the way values and object references are linked across concatenated arrays. In the first group, if you enter a new value to be assigned to arrayThree[0], the new value replaces the string value in the combined array. Because regular values do not maintain a link back to the original array, only the entry in the combined array is changed. A call to showArrays() proves that only the third array is affected by the change.

More complex is the object relationship for this demonstration. A reference to the first text box of the second grouping has been assigned to the third entry of arrayTwo. After concatenation, the same reference is now in the last entry of the combined array. If you enter a new value for a property of the object in the last slot of arrayThree, the change goes all the way back to the original object — the first text box in the lower grouping. Thus, the text of the original field changes in response to the change of arrayThree[5]. And because all references to that object yield the same result, the reference in arrayTwo[2] points to the same text object, yielding the same new answer. The display of the array contents doesn't change, because both arrays still contain a reference to the same object (and the VALUE attribute showing in the <INPUT> tag of the column listings refers to the default value of the tag, not to its current algorithmically retrievable value shown in the last two fields of the page).

Listing 37-6: **Array Concatenation**

```
<HTML>
<HEAD>
<TITLE>Array Concatenation</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
// global variables
var arrayOne, arrayTwo, arrayThree, textObj
// initialize after load to access text object in form
function initialize() {
    var form = document.forms[0]
    textObj = form.original
    arrayOne = new Array("Jerry", "Elaine","Kramer")
    arrayTwo = new Array("Ross", "Rachel",textObj)
    arrayThree = arrayOne.concat(arrayTwo)
    update1(form)
    update2(form)
    showArrays()
}
// display current values of all three arrays
function showArrays() {
    var form = document.forms[0]
    form.array1.value = arrayOne.join("\n")
    form.array2.value = arrayTwo.join("\n")
```

Continued

Listing 37-6 *(continued)*

```
    form.array3.value = arrayThree.join("\n")
}
// change the value of first item in Array Three
function update1(form) {
    arrayThree[0] = form.source1.value
    form.result1.value = arrayOne[0]
    form.result2.value = arrayThree[0]
    showArrays()
}
// change value of object property pointed to in Array Three
function update2(form) {
    arrayThree[5].value = form.source2.value
    form.result3.value = arrayTwo[2].value
    form.result4.value = arrayThree[5].value
    showArrays()
}
</SCRIPT>
</HEAD>
<BODY onLoad="initialize()">
<FORM>
<TABLE>
<TR><TH>arrayOne</TH><TH>arrayTwo</TH><TH>arrayThree</TH></TR>
<TR>
<TD><TEXTAREA NAME="array1" COLS=25 ROWS=6></TEXTAREA></TD>
<TD><TEXTAREA NAME="array2" COLS=25 ROWS=6></TEXTAREA></TD>
<TD><TEXTAREA NAME="array3" COLS=25 ROWS=6></TEXTAREA></TD>
</TR>
</TABLE>
<B>Enter new value for arrayThree[0]:</B><INPUT TYPE="text" NAME="source1"
VALUE="Jerry">
<INPUT TYPE="button" VALUE="Change arrayThree[0]"
onClick="update1(this.form)"><BR>
Current arrayOne[0] is:<INPUT TYPE="text" NAME="result1"><BR>
Current arrayThree[0] is:<INPUT TYPE="text" NAME="result2"><BR>
<HR>

textObj assigned to arrayTwo[2]:<INPUT TYPE="text" NAME="original"
onFocus="this.blur()"></BR>
<B>Enter new value for arrayThree[5]:</B><INPUT TYPE="text" NAME="source2"
VALUE="Phoebe">
<INPUT TYPE="button" VALUE="Change arrayThree[5].value"
onClick="update2(this.form)"><BR>
Current arrayTwo[2].value is:<INPUT TYPE="text" NAME="result3"><BR>
Current arrayThree[5].value is:<INPUT TYPE="text" NAME="result4"><P>

<INPUT TYPE="button" VALUE="Reset" onClick="location.reload()">
</FORM>
</BODY>
</HTML>
```

array.join(*separatorString*)

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓		✓	✓	✓	✓

Example

The script in Listing 37-7 converts an array of planet names into a text string. The page provides you with a field to enter the delimiter string of your choice and shows the results in a textarea.

Listing 37-7: **Using the Array.join() Method**

```
<HTML>
<HEAD>
<TITLE>Array.join()</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
solarSys = new Array(9)
solarSys[0] = "Mercury"
solarSys[1] = "Venus"
solarSys[2] = "Earth"
solarSys[3] = "Mars"
solarSys[4] = "Jupiter"
solarSys[5] = "Saturn"
solarSys[6] = "Uranus"
solarSys[7] = "Neptune"
solarSys[8] = "Pluto"

// join array elements into a string
function convert(form) {
    var delimiter = form.delim.value
    form.output.value = unescape(solarSys.join(delimiter))
}
</SCRIPT>
<BODY>
<H2>Converting arrays to strings</H2>
This document contains an array of planets in our solar system.<HR>
<FORM>
Enter a string to act as a delimiter between entries:
<INPUT TYPE="text" NAME="delim" VALUE="," SIZE=5><P>
<INPUT TYPE="button" VALUE="Display as String" onClick="convert(this.form)">
<INPUT TYPE="reset">
<TEXTAREA NAME="output" ROWS=4 COLS=40 WRAP="virtual">
</TEXTAREA>
</FORM>
</BODY>
</HTML>
```

Notice that this method takes the parameter very literally. If you want to include nonalphanumeric characters, such as a newline or tab, do so with URL-encoded characters (`%0D` for a carriage return; `%09` for a tab) instead of inline string literals. In Listing 37-7, the results of the *`array`*`.join()` method are subjected to the `unescape()` function in order to display them in the TEXTAREA.

`array``.reverse()`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓		✓	✓	✓	✓

Example

Listing 37-8 is an enhanced version of Listing 37-7, which includes another button and function that reverse the array and display it as a string in a text area.

Listing 37-8: **Array.reverse() Method**

```
<HTML>
<HEAD>
<TITLE>Array.reverse()</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
solarSys = new Array(9)
solarSys[0] = "Mercury"
solarSys[1] = "Venus"
solarSys[2] = "Earth"
solarSys[3] = "Mars"
solarSys[4] = "Jupiter"
solarSys[5] = "Saturn"
solarSys[6] = "Uranus"
solarSys[7] = "Neptune"
solarSys[8] = "Pluto"

// show array as currently in memory
function showAsIs(form) {
    var delimiter = form.delim.value
    form.output.value = unescape(solarSys.join(delimiter))
}
// reverse array order, then display as string
function reverseIt(form) {
    var delimiter = form.delim.value
    solarSys.reverse()   // reverses original array
    form.output.value = unescape(solarSys.join(delimiter))
}
</SCRIPT>
<BODY>
<H2>Reversing array element order</H2>
This document contains an array of planets in our solar system.<HR>
<FORM>
```

```
Enter a string to act as a delimiter between entries:
<INPUT TYPE="text" NAME="delim" VALUE="," SIZE=5><P>
<INPUT TYPE="button" VALUE="Array as-is" onClick="showAsIs(this.form)">
<INPUT TYPE="button" VALUE="Reverse the array" onClick="reverseIt(this.form)">
<INPUT TYPE="reset">
<INPUT TYPE="button" VALUE="Reload" onClick="self.location.reload()">
<TEXTAREA NAME="output" ROWS=4 COLS=60>
</TEXTAREA>
</FORM>
</BODY>
</HTML>
```

Notice that the `solarSys.reverse()` method stands by itself (meaning, nothing captures the returned value) because the method modifies the `solarSys` array. You then run the now inverted `solarSys` array through the *`array`*`.join()` method for your text display.

`array``.sort([`*`compareFunction`*`])`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility		✓	✓	✓		✓	✓	✓	✓

Example

You can look to Listing 37-9 for a few examples of sorting an array of string values (see Figure 17-2). Four buttons summon different sorting routines, three of which invoke comparison functions. This listing sorts the planet array alphabetically (forward and backward) by the last character of the planet name and also by the length of the planet name. Each comparison function demonstrates different ways of comparing data sent during a sort.

Listing 37-9: **Array.sort() Possibilities**

```
<HTML>
<HEAD>
<TITLE>Array.sort()</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
solarSys = new Array(9)
solarSys[0] = "Mercury"
solarSys[1] = "Venus"
solarSys[2] = "Earth"
solarSys[3] = "Mars"
solarSys[4] = "Jupiter"
solarSys[5] = "Saturn"
solarSys[6] = "Uranus"
```

Continued

Listing 37-9 *(continued)*

```
solarSys[7] = "Neptune"
solarSys[8] = "Pluto"
// comparison functions
function compare1(a,b) {
    // reverse alphabetical order
    if (a > b) {return -1}
    if (b > a) {return 1}
    return 0
}
function compare2(a,b) {
    // last character of planet names
    var aComp = a.charAt(a.length - 1)
    var bComp = b.charAt(b.length - 1)
    if (aComp < bComp) {return -1}
    if (aComp > bComp) {return 1}
    return 0
}
function compare3(a,b) {
    // length of planet names
    return a.length - b.length
}
// sort and display array
function sortIt(form, compFunc) {
    var delimiter = ";"
    if (compFunc == null) {
        solarSys.sort()
    } else {
        solarSys.sort(compFunc)
    }
    // display results in field
    form.output.value = unescape(solarSys.join(delimiter))
}
</SCRIPT>
<BODY onLoad="document.forms[0].output.value = unescape(solarSys.join(';'))">
<H2>Sorting array elements</H2>
This document contains an array of planets in our solar system.<HR>
<FORM>
Click on a button to sort the array:<P>
<INPUT TYPE="button" VALUE="Alphabetical A-Z" onClick="sortIt(this.form)">
<INPUT TYPE="button" VALUE="Alphabetical Z-A"
onClick="sortIt(this.form,compare1)">
<INPUT TYPE="button" VALUE="Last Character"
onClick="sortIt(this.form,compare2)">
<INPUT TYPE="button" VALUE="Name Length" onClick="sortIt(this.form,compare3)">
<INPUT TYPE="button" VALUE="Reload Original" onClick="self.location.reload()">
<INPUT TYPE="text" NAME="output" SIZE=62>
</TEXTAREA>
</FORM>
</BODY>
</HTML>
```

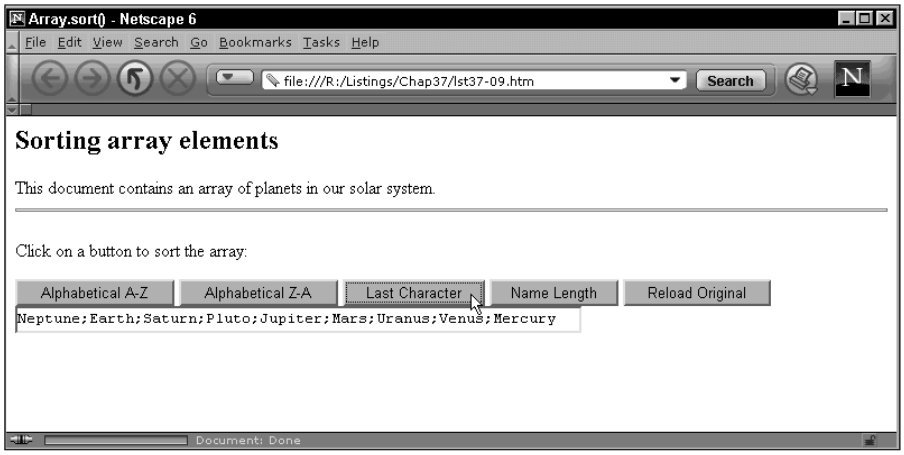


Figure 17-2: Sorting an array of planet names alphabetically by last character

array.splice(*startIndex* , *deleteCount*[, *item1*[, *item2*[,...*itemN*]]])

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓	✓					✓

Example

Use The Evaluator (Chapter 13 in the *JavaScript Bible*) to experiment with the `splice()` method. Begin by creating an array with a sequence of numbers:

```
a = new Array(1,2,3,4,5)
```

Next, remove the center three items, and replace them with one string item:

```
a.splice(1, 3, "two/three/four")
```

The Results box shows a string version of the three-item array returned by the method. To view the current contents of the array, enter `a` into the top text box.

To put the original numbers back into the array, swap the string item with three numeric items:

```
a.splice(1, 1, 2, 3, 4)
```

The method returns the single string, and the `a` array now has five items in it again.

✦ ✦ ✦

APPENDIX

What's on the CD-ROM

The accompanying Windows–Macintosh CD-ROM contains a complete set of HTML document listings and an electronic version of this book, plus additional listings and the full text of the *JavaScript Bible, Gold Edition*. You also receive Adobe Acrobat Reader software to view and search the electronic versions of the books.

System Requirements

To derive the most benefit from the example listings, you should have both Netscape Navigator 6 (or later) and Internet Explorer 5 (or later) installed on your computer. While many scripts run in both browsers, several scripts demonstrate features that are available on only one browser or the other. To write scripts, you can use a simple text editor, word processor, or dedicated HTML editor.

To use the Adobe Acrobat Reader, you need the following:

- For Windows 95, Windows 98, or Windows NT 4.0 (with SP3 or later), you should be using a 486 or Pentium computer with 16MB of RAM and 10MB of hard disk space.
- Macintosh users require a PowerPC, System 7.1,2 or later, at least 8MB of RAM, and 8MB of disk space.

Disc Contents

Platform-specific software is located in the appropriate Windows and Macintosh directories on the CD-ROM. The contents include the following items.

JavaScript listings for Windows and Macintosh text editors

Almost all example listings from this book and the *JavaScript Bible, Gold Edition* are on the CD-ROM in the form of complete HTML files, which you can load into a browser to see the language item in operation (a few others are plain text

files, which you can view in your browser or text editor). A directory called Listings contains the example files, with nested folders named for each chapter of the *JavaScript Bible*. Each HTML file's name is keyed to the Listing number in the book. For example, the file for Listing 15-1 is named `lst15-01.htm`. Note that the first part of each listing number is keyed to a *JavaScript Bible* chapter number. Thus, Listing 15-1 demonstrates a term discussed in Chapter 15 of the *JavaScript Bible* (both editions), although the printed listing and discussion about the listing appears in Chapter 1 of this book because Chapter 1 contains examples for *JavaScript Bible* Chapter 15.

For your convenience, an `index.html` file in the Listings folder provides a front-end table of contents to the HTML files for the book's program listings. Open that file from your browser whenever you want to access the program listing files. If you intend to access that index page frequently, you can bookmark it in your browser(s). Using the index file to access the listing files can be very important in some cases, because several individual files must be opened within their associated framesets to work properly. Accessing the files through the `index.html` file assures that you open the frameset. The `index.html` file also shows browser compatibility ratings for all the listings. This saves you time from opening listings that are not intended to run on your browser. To examine and modify the HTML source files, open them from your favorite text editor program (for Windows editors, be sure to specify the `.htm` file extension in the Open File dialog box).

You can open all example files directly from the CD-ROM, but if you copy them to your hard drive, access is faster and you will be able to experiment with modifying the files more readily. Copy the folder named Listings from the CD-ROM to any location on your hard drive.

Electronic versions of the books

These are complete, searchable versions of both this book and the *JavaScript Bible, Gold Edition*, provided in Adobe Acrobat `.pdf` format. The Acrobat text for this book is in the folder named JSExamples PDF, while the *JavaScript Bible* text is in the JSBGold PDF folder.

Adobe Acrobat Reader

The Adobe Acrobat Reader is a helpful program that enables you to view the entire contents of both this book and the *JavaScript Bible, Gold Edition*, which are in `.pdf` format on the CD-ROM. To install and run Adobe Acrobat Reader, follow these steps:

For Windows

1. Start Windows Explorer or Windows NT Explorer and then open the Acrobat folder on the CD-ROM.
2. In the Acrobat folder, double-click the rs405eng.exe icon and follow the instructions presented on-screen for installing Adobe Acrobat Reader.

For Macintosh

1. Open the Acrobat folder on the CD-ROM.
2. In the Acrobat folder, double-click the Adobe Acrobat Installer icon and follow the instructions presented on-screen for installing Adobe Acrobat Reader.

✦ ✦ ✦

Index

C

D

E

Continued

F

Continued

G

H

I

K

L

M

Continued

N

O

P

Continued

Continued

S

T

U

V

W

X

Y

Z

Hungry Minds, Inc.
End-User License Agreement

READ THIS. You should carefully read these terms and conditions before opening the software packet(s) included with this book ("Book"). This is a license agreement ("Agreement") between you and Hungry Minds, Inc. ("HMI"). By opening the accompanying software packet(s), you acknowledge that you have read and accept the following terms and conditions. If you do not agree and do not want to be bound by such terms and conditions, promptly return the Book and the unopened software packet(s) to the place you obtained them for a full refund.

1. **License Grant.** HMI grants to you (either an individual or entity) a nonexclusive license to use one copy of the enclosed software program(s) (collectively, the "Software") solely for your own personal or business purposes on a single computer (whether a standard computer or a workstation component of a multi-user network). The Software is in use on a computer when it is loaded into temporary memory (RAM) or installed into permanent memory (hard disk, CD-ROM, or other storage device). HMI reserves all rights not expressly granted herein.
2. **Ownership.** HMI is the owner of all right, title, and interest, including copyright, in and to the compilation of the Software recorded on the disk(s) or CD-ROM ("Software Media"). Copyright to the individual programs recorded on the Software Media is owned by the author or other authorized copyright owner of each program. Ownership of the Software and all proprietary rights relating thereto remain with HMI and its licensers.
3. **Restrictions On Use and Transfer.**
   - **(a)** You may only (i) make one copy of the Software for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided that you keep the original for backup or archival purposes. You may not (i) rent or lease the Software, (ii) copy or reproduce the Software through a LAN or other network system or through any computer subscriber system or bulletin-board system, or (iii) modify, adapt, or create derivative works based on the Software.
   - **(b)** You may not reverse engineer, decompile, or disassemble the Software. You may transfer the Software and user documentation on a permanent basis, provided that the transferee agrees to accept the terms and conditions of this Agreement and you retain no copies. If the Software is an update or has been updated, any transfer must include the most recent update and all prior versions.

4. **Restrictions on Use of Individual Programs.** You must follow the individual requirements and restrictions detailed for each individual program in the Appendix of this Book. These limitations are also contained in the individual license agreements recorded on the Software Media. These limitations may include a requirement that after using the program for a specified period of time, the user must pay a registration fee or discontinue use. By opening the Software packet(s), you will be agreeing to abide by the licenses and restrictions for these individual programs that are detailed in the Appendix and on the Software Media. None of the material on this Software Media or listed in this Book may ever be redistributed, in original or modified form, for commercial purposes.

5. **Limited Warranty.**

   (a) HMI warrants that the Software and Software Media are free from defects in materials and workmanship under normal use for a period of sixty (60) days from the date of purchase of this Book. If HMI receives notification within the warranty period of defects in materials or workmanship, HMI will replace the defective Software Media.

   (b) **HMI AND THE AUTHOR OF THE BOOK DISCLAIM ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE PROGRAMS, THE SOURCE CODE CONTAINED THEREIN, AND/OR THE TECHNIQUES DESCRIBED IN THIS BOOK. HMI DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE.**

   (c) This limited warranty gives you specific legal rights, and you may have other rights that vary from jurisdiction to jurisdiction.

6. **Remedies.**

   (a) HMI's entire liability and your exclusive remedy for defects in materials and workmanship shall be limited to replacement of the Software Media, which may be returned to HMI with a copy of your receipt at the following address: Software Media Fulfillment Department, Attn.: *JavaScript Examples Bible: The Essential Companion to JavaScript Bible*, Hungry Minds, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, or call 1-800-762-2974. Please allow four to six weeks for delivery. This Limited Warranty is void if failure of the Software Media has resulted from accident, abuse, or misapplication. Any replacement Software Media will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

(b) In no event shall HMI or the author be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising from the use of or inability to use the Book or the Software, even if HMI has been advised of the possibility of such damages.

(c) Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation or exclusion may not apply to you.

7. **U.S. Government Restricted Rights.** Use, duplication, or disclosure of the Software for or on behalf of the United States of America, its agencies and/or instrumentalities (the "U.S. Government") is subject to restrictions as stated in paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013, or subparagraphs (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR supplement, as applicable.

8. **General.** This Agreement constitutes the entire understanding of the parties and revokes and supersedes all prior agreements, oral or written, between them and may not be modified or amended except in a writing signed by both parties hereto that specifically refers to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict herewith. If any one or more provisions contained in this Agreement are held by any court or tribunal to be invalid, illegal, or otherwise unenforceable, each and every other provision shall remain in full force and effect.

CD-ROM Installation Instructions

The files on this CD-ROM can be accessed and used from both Windows 95 (or later) and Macintosh environments. Some Macintosh program files require MacOS 8.6 or later, but program listing text files can be opened with any MacOS version. For Windows, access the software with My Computer or Windows Explorer. Macintosh users can access files by using the Finder.

You can open all of the example file listings directly from the CD-ROM, but access will be faster — and you will be able to experiment with modifying the files more readily — if you copy the listings to your hard drive. Copy the folder named Listings from the CD-ROM to any location on your hard drive.

To open the listing scripts on this CD-ROM, you should have a copy of Microsoft Internet Explorer 5 (or later), Netscape Navigator 6 (or later), or both browsers installed on your computer.

To run the listing scripts from your browser, open the file named index.html in the Listings folder. This page provides a table of contents consisting of direct links to the listings, showing which browsers are compatible with each listing.

Access the Adobe Acrobat (PDF) files for the book's contents from the CD-ROM. Be sure to install the index files into your copy of Acrobat to take advantage of full-text search.

For more details on installing and running the CD-ROM contents, see the Appendix.