

G U Í A
E S E N C I A L

JAVASCRIPT



- Obtenga resultados inmediatos con JavaScript.
- Aprenda con ejemplos prácticos reales.
- Cada técnica clave se explica de forma rápida, clara y sencilla.
- Aprenda a detectar el navegador, crear contenido dinámico, *rollovers* de imágenes, técnicas de navegación y mucho más.
- Escrito por los mejores profesionales de la Web.

Prentice
Hall

DAN BARRETT
DAN LIVINGSTON
MICAH BROWN

005.13
B.274

Guía esencial JavaScript

Consultores editoriales:

SEBASTIÁN DORMIDO BENCOMO

Departamento de Informática y Automática

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

LUIS JOYANES AGUILAR

Departamento de Lenguajes, Sistemas Informáticos e

Ingeniería del Software

UNIVERSIDAD PONTIFICIA DE SALAMANCA en Madrid

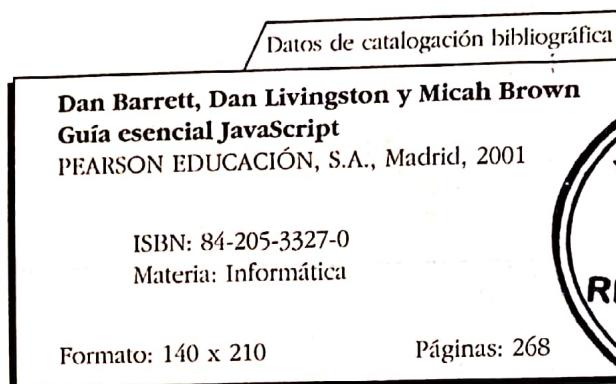
Guía esencial JavaScript

**Dan Barrett
Dan Livingston
Micah Brown**

**Traducción
*Sistemas KME, S.L.***



Madrid • México • Santafé de Bogotá • Buenos Aires • Caracas • Lima
Montevideo • San Juan • San José • Santiago • Sao Paulo • White Plains



Dan Barrett, Dan Livingston y Micah Brown
Guía esencial JavaScript

No está permitida la reproducción total o parcial de esta obra ni su tratamiento o transmisión por cualquier medio o método sin autorización escrita de la Editorial.

DERECHOS RESERVADOS

© 2001 respecto a la primera edición en español por:

PEARSON EDUCACIÓN, S.A.

Núñez de Balboa, 120
28006 Madrid

ISBN: 84-205-3327-0

Depósito Legal: M- 25.632-2001

PRENTICE HALL es un sello editorial autorizado de PEARSON EDUCACIÓN

Traducido de: Essential JavaScript for Web Professionals

Copyright © 2000 by Prentice Hall PTR

ISBN: 0-13-013056-7

Editores de la edición en español: Alejandro Domínguez

Félix Fuentes

Eva María López

Cubierta: Mario Guindel y Yann Boix

Composición: DiScript Preimpresión, S.L.

Impreso por: Gráficas Rógar, S. A.

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

Este libro ha sido impreso con papel y tintas ecológicos

Índice de contenido

<i>Introducción</i>	ix	
<i>Reconocimientos</i>	xv	
<i>Sobre los autores</i>	xix	
Capítulo 1	Dinamismo y detección	1
Proyecto I: cómo generar el contenido específico de la plataforma	2	
Insertar un <i>script</i> en el HTML	3	
Jerarquías en JavaScript	5	
Detección de plataforma/navegador	6	
Crear HTML dinámico desde dentro de JavaScript	13	
Proyecto II: cómo imprimir información de <i>copyright</i> y la fecha de la última modificación	16	
Iniciación a la propiedad LastModified	17	
Imprimir dinámicamente datos que no son cadenas de caracteres	20	
Resumen	24	
Proyectos avanzados	25	

Capítulo 2

Rollovers de imagen	27
Proyecto I: <i>script</i> del <i>rollover</i>	29
Creación de los objetos IMAGE	29
Funciones de <i>rollovers</i> de imagen	37
Inserción de los manipuladores de eventos	41
Proyecto II: cómo añadir funcionalidad avanzada a los <i>rollovers</i>	47
Inserción de los manipuladores de eventos	49
Creación de los objetos IMAGE	52
Inserción de las funciones del <i>rollover</i>	54
Peculiaridades de los <i>rollovers</i>	66
Resumen	67
Proyectos avanzados	68

Capítulo 3

JavaScript para navegación	69
Proyecto I: JavaScript y los menús desplegables	70
Creación de la función de navegación	71
Inserción del manipulador de eventos	73
Uso de onChange para una gratificación instantánea	74
Proyecto II: cómo utilizar varios menús desplegables para navegar	79
Cómo crear los <i>arrays</i>	80
Creación de la función de población del menú	85
Creación de la función de navegación	99
Proyecto III: cómo utilizar JavaScript en una página de inicio de sesión	108

Cómo insertar el manipulador
de eventos 110

Resumen 111

Proyectos avanzados 111

Capítulo 4

Manipulación de errores en los formularios 113

Proyecto I: cómo comprobar los campos
de formulario vacíos 114

Comprobación de formularios en busca
de campos de texto vacíos 119

Cómo asegurarse de que se selecciona
una opción del menú desplegable 124

Proyecto II: cómo comprobar datos
erróneos en los campos de un
formulario 126

Comprobación de caracteres correctos
en las direcciones de correo electrónico 126

Comprobación de la existencia del prefijo
en los números telefónicos 128

Cómo asegurarse de que los códigos postales
sólo tienen números 131

Proyecto III: cómo permitir que el usuario
sepa qué es erróneo 140

Resumen 149

Proyectos avanzados 149

Capítulo 5

Ventanas y marcos JavaScript 151

Una mirada al objeto WINDOW 152

Proyecto I: cómo crear, llenar y cerrar
ventanas 152

Creación y definición de arrays 153

Creación de la función 156

Capítulo 6**JavaScript avanzado** 183

Proyecto I: cómo crear submenús para las páginas secundarias de Stitch 183

 Especificaciones de funcionalidad 184

 Creación y población de los *arrays* con los datos de almacenamiento 189

 Creación de las funciones para ejecutar el sistema de submenús 193

 Inserción de los manipuladores de eventos 206

 Resumen 215

 Proyectos avanzados 216

Apéndice A**Manipuladores de eventos** 217**Apéndice B****Objetos de JavaScript** 225**Índice alfabético** 245

Insertar los manipuladores de eventos 162

Proyecto II: cómo utilizar JavaScript para comunicarse entre marcos 168

 Desentrañando la jerarquía de marcos 168

 Creación de una función para comprobar en qué sección está 171

 Inserción del manipulador de eventos 175

 Resumen 181

 Proyectos avanzados 181

Introducción

¡Bienvenido! Este libro es algo que ojalá hubiésemos tenido cuando comenzamos con JavaScript. En aquel tiempo, había básicamente dos tipos de manuales en el mercado: tomos de 1.200 páginas de conocimiento aparentemente arcano, y libros demasiado simplificados y carentes de información práctica. Desafortunadamente, no había libros que fuesen informativos y al mismo tiempo proporcionasen una enseñanza que pudiera ser utilizada rápida y efectivamente en situaciones del mundo real.

Este libro le guiará a través de JavaScript utilizando ejemplos tomados directamente de situaciones con las que nos encontramos todos los días durante la construcción de sitios web. Se inicia con ejemplos sencillos y se vuelve bastante sofisticado con el scripting hacia el final del libro. Dicho esto, echemos un vistazo a cómo está configurado este libro y un breve resumen de *scripting*, así como qué puede hacer con JavaScript y qué no.

Cómo está configurado este libro

Lo más probable es que haya cogido este libro cuando el jefe le llamó a la oficina y le enseñó un sitio web que hacía uso de JavaScript. Se le dijo sin dejar lugar a dudas que el

trabajo sería implementar la misma característica, o similar, en el sitio web. “Ningún problema” fue su respuesta, mientras se decía a sí mismo: “¡Mejor aprendo JavaScript... y rápido!”.

Así es como a menudo desarrollamos nuestras habilidades: nos dan un trabajo y, si no sabemos exactamente cómo hacerlo, enseguida aprendemos cómo. Siguiendo con este modelo real, este libro está dividido en dos proyectos principales. En cada uno de los proyectos, seremos responsables de crear y/o actualizar el sitio web de una compañía ficticia.

Durante los tres primeros capítulos, renovaremos la página de inicio de Shelley Biotechnologies, una *startup* biotecnológica de rápido crecimiento. En cada capítulo tenemos al menos un proyecto que consta de soluciones JavaScript utilizadas habitualmente cuya dificultad varía de fácil a moderada. Al final de cada capítulo hay más ejercicios avanzados que puede completar por su cuenta para desarrollar sus habilidades. En la segunda mitad del libro añadiremos muchas más cosas necesarias para el sitio web de Stitch Magazine. Los ejemplos serán más avanzados que los del primer proyecto y mostrarán algunas de las cosas que puede hacer utilizando JavaScript.

Los ejercicios de los capítulos están diseñados para darle una sólida base en JavaScript sobre la que construir a medida que sigue utilizándolo. Encontrará que en la mayoría de las ocasiones hay más de una forma de hacer las cosas en JavaScript, realmente no hay formas correctas o erróneas de realizar tareas.

En el área de descarga de la sección de informática de Pearson Educación (www.pearson.es) puede descargar un archivo zip con los archivos HTML y las imágenes necesarias para realizar los ejercicios.

Una introducción a JavaScript

¿Qué es JavaScript?

Para los que sean nuevos en el mundo del desarrollo de la Web y que tal vez estén aprendiendo JavaScript junto con HTML, puede ser necesaria una breve introducción a JavaScript. JavaScript es el lenguaje de *script*, multiplataforma incluido en Netscape. Como HTML, funcionará sobre todas las plataformas.

JavaScript le permite ampliar la funcionalidad de las páginas web insertando aplicaciones directamente en el código HTML. Puede utilizar JavaScript para construir aplicaciones que varían desde añadir interactividad a las páginas hasta aplicaciones que interactúan con bases de datos. Aunque Netscape creó JavaScript, funcionará con la mayoría de los navegadores modernos, incluyendo Internet Explorer (IE) de Microsoft. Sin embargo, IE no soporta directamente JavaScript. IE tiene su propio lenguaje de *script*, Jscript, que soporta la mayoría de las características encontradas en JavaScript. En los pocos casos en los que los lenguajes difieren, se destacan dichas diferencias y se presenta una forma de evitarlas. Como esos son los dos principales navegadores del mercado, los *scripts* que escribiremos se centrarán en ellos.

Hay dos métodos que puede utilizar para incluir JavaScript en las páginas web, del lado del cliente y del lado del servidor.

Ambos métodos comparten el mismo conjunto de fundamentos del lenguaje. El núcleo de este lenguaje define un conjunto base de objetos y características que funcionarán tanto en aplicaciones cliente como servidor.

Cada método tiene su propio conjunto de características y objetos extendidos.

JavaScript del lado del cliente: cómo funciona

Las aplicaciones cliente de JavaScript son *scripts* que se integran directamente en las páginas HTML y que son ejecutadas por el navegador del usuario cuando carga la página. Al menos el 90% de los *scripts* que encuentre en la Web están dentro de esta categoría. Por tanto, éste es el método que utilizaremos a lo largo de este libro.

Cuando el navegador del usuario carga una página HTML con JavaScript insertado en ella, el motor de ejecución JavaScript del navegador interpreta el *script* de arriba abajo, ejecutando las sentencias según desciende.

Una de las ventajas del uso de *scripts* del lado del cliente es que el *script* puede detectar y hacer uso de eventos iniciados por el usuario, como cambios en un formulario o el ratón moviéndose sobre un objeto gráfico determinado. El *script* es entonces capaz de utilizar esa información para llamar a otras partes del *script*, y todo esto puede hacerse sin volver al servidor web y obtener más información. Dado que nuestros *scripts* son dependientes de ser interpretados por el navegador del usuario, unas pocas palabras sobre los diferentes navegadores y cómo difieren en el manejo de JavaScript nos vendrán bien.

Navegadores y versiones

Como se dijo anteriormente, IE y Navigator difieren ligeramente en la implementación de los lenguajes de *script*. Como programador, esto hace su vida un poco más difícil: habrá ocasiones en las que una solución funcionará de modo diferente o no en los distintos navegadores. Esperar puede ser peor: al sacar Netscape y Microsoft nuevas versiones de los navegadores, las versiones de los lenguajes de *script* también cambian. Esto significa que los *scripts* escritos utilizando nuevas características pueden no funcionar en un

navegador más antiguo. Pero no se disguste, no es tan malo como parece. Todo esto significa que tendrá que tener un poco más de cuidado al escribir y verificar los *scripts*. Hay muchas técnicas que puede utilizar para asegurarse de que los *scripts* funcionarán en general, exploraremos esas técnicas y las ocasiones apropiadas para utilizarlas. Sin embargo, como este libro tiene JavaScript en el título en lugar de JScript, nos concentraremos principalmente sobre el lenguaje de *script* de Netscape.

Qué puede hacer JavaScript y qué no

Mientras las aplicaciones que pueda crear utilizando JavaScript sólo están limitadas por su imaginación, hay varias cosas que no puede hacer, como acceder o controlar la computadora del usuario. Por razones de seguridad, la escritura en la computadora del usuario está severamente limitada. Puede almacenar datos en la computadora del usuario mediante el uso de una *cookie*, e incluso entonces estará limitado a un simple fichero de texto. Esto protege a los usuarios de *scripts* que podrían dañar sus computadoras o permitir a programadores sin escrúpulos acceder a información personal.

Una característica llamada “Política del mismo origen” también restringe el acceso de los *scripts* de un origen a ciertas propiedades o archivos de otras ubicaciones. Por ejemplo, si tiene un *script* ubicado en <http://www.your-site.com/test.html> e intenta acceder a ciertas propiedades de una página HTML situada en <http://www.theirsite.com/test.html>, la Política del mismo origen denegará el acceso del *script*. Las propiedades que la Política del mismo origen restringe se muestran en la Figura I.1.

Éstas son las principales restricciones que encontrará al escribir aplicaciones JavaScript. Estamos seguros de que se

Objetos	Propiedades restringidas
images	src, lowsrc
document	anchors, applets, cookie, domain, elements, embeds, forms, lastModified, length, links, referrer, title, URI
layer	src
location	Todas las propiedades excepto para x e y
window	find

Figura I.1

Restricciones de la Política del mismo origen.

encontrará en ocasiones intentando utilizar un objeto o propiedad para hacer algo que no se pueda, pero dichas limitaciones pueden considerarse menos restrictivas que hecho de aprender la estructura del lenguaje.

Mientras comienza, si piensa en una posible solución puede diferir de los ejemplos de este libro, inténtelo menudo puede tropezar con una solución en la que otros hayan pensado. Dicho esto, comencemos con el aprendizaje.

Reconocimientos

Dan Barrett

Me gustaría agradecer a la Academia... Un momento, me he confundido. Ya en serio, me gustaría agradecer a mi esposa Kristin su apoyo y comprensión mientras escribía este libro.

Gracias a mi familia, mis padres Chuck y Judy, y mi hermano Rick, quienes me animaron a meter mis narices entre los libros durante la mayor parte de mi infancia. También me gustaría agradecer a Dan Livingston y a Micah Brown que pensasen en mí cuando se realizó este proyecto, y a Brad Scott por dar lo mejor de sí mismo en la revisión técnica.

Finalmente, este libro no se hubiese podido escribir sin la maravillosa cafeína que es Mountain Dew.

Dan Livingston

Hemos tenido la maravillosa suerte de poder trabajar con Mark Taub y Karen McLean, de Prentice Hall, en este proyecto. Hemos probado especialmente el aguante de Karen, que siempre se mostró paciente y colaboradora durante el proceso.

Me gustaría agradecer a mi novia, Tanya Muller, su continua paciencia y estímulo. No hubiera podido escribir este libro si no hubiese estado siempre a mi lado. Su apoyo y continúa siendo, inestimable.

También me gustaría agradecer a W. Bradley Scott, Clear Ink, por contarme la idea de utilizar una revista moda *on-line* como empresa ficticia. También ha actuado como revisor técnico, y su ayuda siempre ha sido buena.

Finalmente, me gustaría dar las gracias a mi guía en diseño, Brad Eigen, de MadBoy Productions.

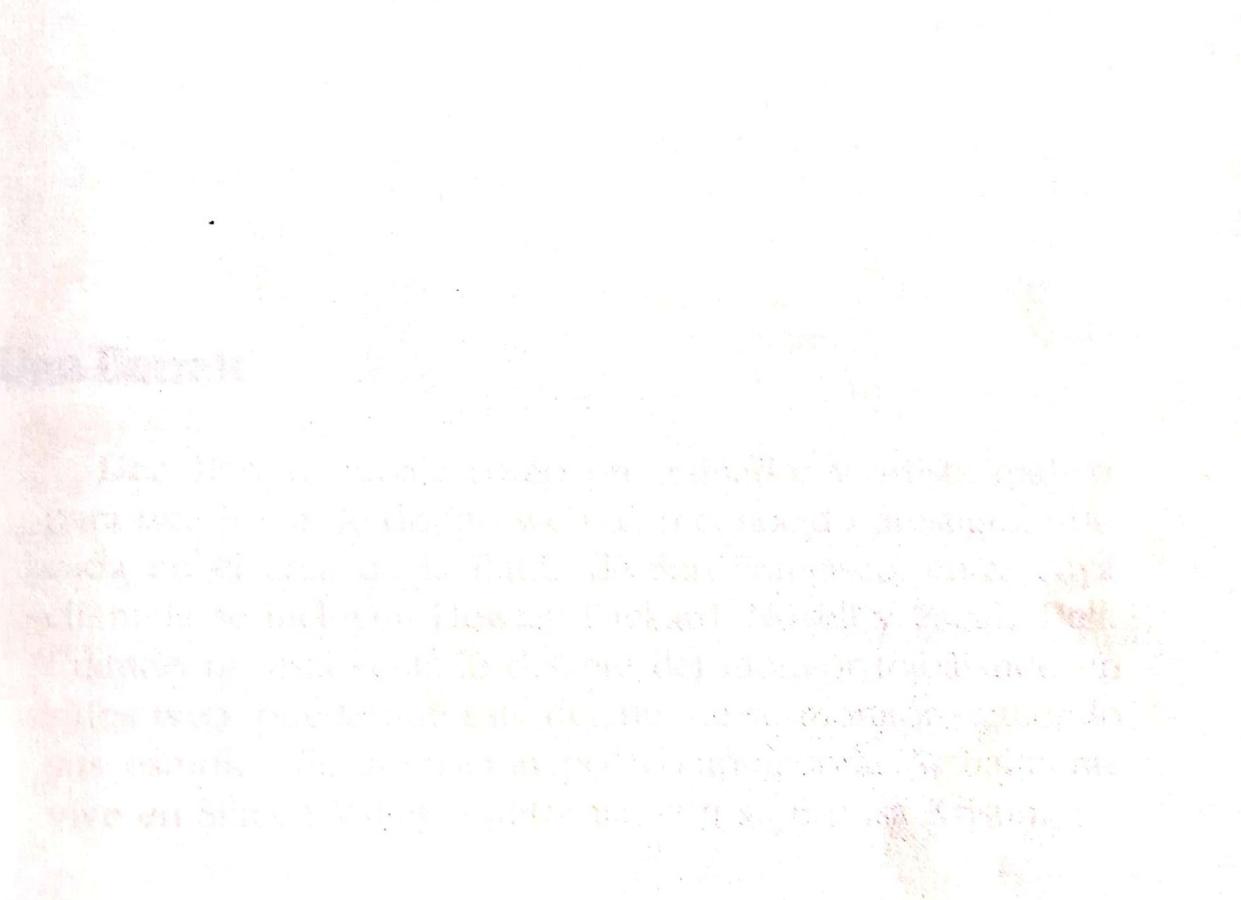
Micah Brown

Me gustaría agradecer de una manera muy especial a mi esposa, Dawn, el que me haya ayudado de todas las maneras posibles. Eres mi amor, mi vida y, lo más importante, mi mejor amiga. Le dedico este libro a ella y a nuestra hija, Ally Nova, que todavía tiene que nacer (estamos deseando conocerte).

También me gustaría agradecer especialmente a mis padres William y Donna, y a mis suegros, Beppe y Joy, todo lo que han hecho por mí estos últimos 29 años. No sería que hoy soy si no fuese por ellos. Les estaré eternamente agradecido por todo lo que me han enseñado y por enseñarme a crecer como persona.

Gracias a Mark Taub y a Karen McLean por ayudarnos. Dan y a mí, a plasmar en este libro lo que antes estaba en nuestras cabezas. Tenéis razón, esto es más duro de lo que imaginaba. También, gracias a Carl Gorman, mi compañero de Etail Enterprises (www.etail.com), y a los miembros de mi banda, Kelly y Carl, de Nitrus, por animarme mientras duró todo esto.

Finalmente, gracias a mis coautores, Dan Barrett y Dan Livingston, así como a todos los que hayan trabajado en los libros de esta serie. Si no fuese por vosotros, no hubiese escrito esto.



Dan Livingston

Dan Livingston es un programador web y autor de libros de programación. Es el autor de la serie de libros *JavaScript para principiantes*, que incluye *JavaScript para principiantes*, *JavaScript para principiantes II*, *JavaScript para principiantes III* y *JavaScript para principiantes IV*. Dan es el autor de *JavaScript para principiantes* y *JavaScript para principiantes II* en la serie *Guías esenciales* de la editorial Packt Publishing. Dan es un miembro activo de la comunidad de desarrollo web y ha contribuido a numerosos proyectos abiertos, entre ellos el popular framework de desarrollo web Node.js. Dan es un conferenciante regular en eventos de desarrollo web y ha impartido numerosas charlas en conferencias internacionales como el FOSDEM y el BarCamp. Dan es un escritor prolífico y ha publicado numerosos artículos y tutoriales en sitios web como [danlivingston.com](http://www.danlivingston.com) y [nodejs.org](http://www.nodejs.org).

Sobre los autores

Dan Barrett

Dan Barrett trabaja como programador y artista gráfico para una firma de diseño web de reconocido prestigio, ubicada en el área de la Bahía de San Francisco, entre cuya clientela se incluyen Hewlett-Packard, Novell y Pacific Bell. Cuando no está sentado delante del monitor trabajando en sitios web, puede que esté delante de su monitor siguiendo sus estudios de animación por computadora. Actualmente vive en Silicon Valley, California, con su esposa Kristin.

Dan Livingston

Dan Livingston desarrolla diseños para la Web desde principios de 1996, con unos escenarios basados en la biología marina. Desde entonces, ha trabajado para clientes tan poderosos como Apple, Pacific Bell y Novell. Sus sitios han ganado numerosos premios y han figurado tanto en libros de diseño, como en la hora de máxima de audiencia de la CNN. Su sitio DHTML, Palette Man, ha recibido el reconocimiento internacional y varios premios "Cool Site" de Yahoo!, Macromedia y USA Today. Dan fue diseñador web y desarrollador de *scripts* de la empresa de diseño Clear Ink, antes

de crear con éxito su propia compañía de diseño, Wires Productions.

Micah Brown

Después de trabajar durante varios años en la industria de la impresión, Micah Brown comenzó su carrera como programador y diseñador en la industria web en 1995. Algunos de los sitios que Micah ha creado fueron para Dr. Pepper Pacific Bell, Amazing Discoveries y Ascend Communications. Micah ha sido también revisor técnico para varias publicaciones de Prentice Hall los últimos tres años, siendo la más notable Perl, de Example, escrita por Ellie Quigley. Actualmente, Micah es copropietario de Etail Enterprise una consultoría web ubicada en el sur de California, que está especializada en acercar a las empresas a esta nueva era de anuncios *on-line*.

Dinamismo y detección

En este capítulo

- Proyecto I: cómo generar contenido específico de la plataforma.
- Proyecto II: cómo imprimir información de *copyright* y la fecha de la última modificación.
- Resumen.
- Proyectos avanzados.

Bien, hemos abordado el trabajo de programar el sitio web para Shelley Biotechnologies. Actualmente, el sitio es muy estático y el nuevo jefe le ha planteado la posibilidad de incrementar la velocidad respecto a la competencia. Sin embargo, antes de comenzar con el trabajo, el jefe quiere que se resuelva un asunto sugerido por alguien que ha estado visitando el sitio. Parece ser que los usuarios con computadoras Macintosh no visualizan el texto de la misma forma que los que navegan por el sitio empleando una computadora basada en Windows (véase la Figura 1.1). Tras una investigación más profunda averiguamos que el tamaño de la fuente se muestra de forma diferente en las dos plataformas, e incluso aunque el problema no es el fin del mundo, el jefe es muy detallista. Éste parece el lugar perfecto para comenzar a utilizar JavaScript.

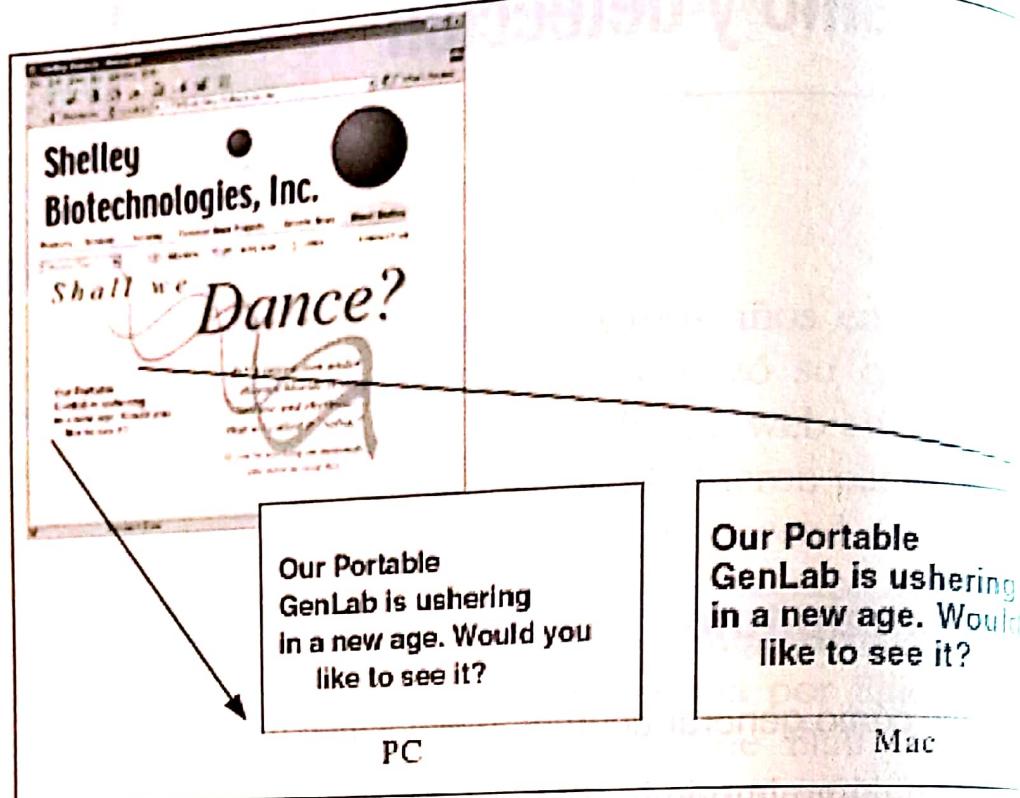


Figura 1.1
Diferencias de fuente sobre plataformas diferentes.

Al utilizar JavaScript seremos capaces de ver qué forma y qué navegador están siendo utilizados para visitar el sitio. Ahora enviaremos el contenido que mejor adapte a cada usuario.

Proyecto 1: cómo generar el contenido específico de la plataforma

Después de juguetear con el código HTML existente llega a la conclusión de que, para que la página se muestre de manera idéntica en ambas plataformas, se requiere una fuente 2 para Macintosh y 3 para Windows. Para resolver el problema vamos a necesitar escribir la etiqueta FONT dinámicamente en el HTML, por lo que el script necesitará estar situado dentro del HTML en el punto donde normalmente estaría situada la etiqueta FONT.

Para este primer *script* hay dos aspectos principales: primero, debemos averiguar qué plataforma está usando el usuario; y segundo, necesitaremos que el navegador imprima dinámicamente el código necesario para cada una.

Insertar un script en el HTML

El primer paso para escribir cualquier cosa en JavaScript es indicar al navegador que el código que está introduciendo en la página web no es HTML, sino una aplicación JavaScript. Una vez sabido esto, enviará la información al motor de ejecución de JavaScript, que ejecutará el *script*. Para la mayoría de *scripts* haremos esto encerrándolos dentro de la etiqueta <SCRIPT>.

```
<SCRIPT>
...
</SCRIPT>
```

Hay otras formas mediante las que insertar los *scripts* en HTML:

- Especificar un fichero externo, que contenga los *scripts*.
- Especificar una expresión JavaScript como valor para un atributo HTML.
- Insertar los *scripts* dentro de algunas etiquetas HTML como manejadores de eventos.

La etiqueta <SCRIPT> es con diferencia el método más utilizado. Repasaremos algunos otros más adelante en el libro cuando comencemos a utilizar JavaScript con imágenes y formularios. Si el navegador encuentra JavaScript fuera de una etiqueta <SCRIPT> o no utilizado según uno de los otros métodos mencionados previamente, el *script* será tratado como texto para ser visualizado en pantalla y no fun-

cionará. Según van saliendo nuevas versiones de JavaScript, hay propiedades y expresiones que son añadidas a las herramientas del lenguaje disponibles. Si utiliza esas nuevas características en los *scripts* y un sistema provisto de un navegador antiguo al cargar el *script*, lo más probable será que le dé un error y que el *script* no funcione correctamente. Para comprobarlo, podemos añadir un atributo a la etiqueta **<SCRIPT>** para indicarle al navegador qué versión de JavaScript estamos codificando. Cuando el navegador usuario llega al *script*, comprobará si soporta la versión especificada y, si no, saltará el *script*. Hacemos esto utilizando el atributo **LANGUAGE** de la etiqueta **<SCRIPT>**.

```
<SCRIPT LANGUAGE="JavaScript1.2">  
...  
</SCRIPT>
```

La adición a la sentencia anterior indica al navegador que si no soporta la versión 1.2. de JavaScript, no debería continuar con el *script*. Este método comprobará que los navegadores soportan JavaScript; sin embargo, es posible que algunos usuarios utilicen aún navegadores antiguos que no soportan JavaScript en absoluto. Para resolver esto podemos poner todo el código HTML entre etiquetas de comentario así:

```
<SCRIPT LANGUAGE="JavaScript">  
<!-- El código tras esto será ignorado por los navegadores<br/>→ más antiguos  
...  
// El código deja de ocultarse aquí. -->  
</SCRIPT>
```

Ahora sabemos cómo poner los *scripts* en el código HTML. Sin embargo, antes de entrar en el script más profundamente, es necesario comprender la jerarquía de JavaScript.

Jerarquías en JavaScript

Cuando el navegador carga una página HTML, el motor de JavaScript crea automáticamente un conjunto de objetos basados en el contenido del código HTML.

Mantiene dichos objetos en una jerarquía, que puede ser utilizada para llamar o hacer referencia a los objetos y sus propiedades (véase la Figura 1.2).

El objeto **WINDOW** está en lo más alto de la jerarquía JavaScript; este objeto es la ventana real en la que aparece el navegador. Los descendientes del objeto **WINDOW** son sus propiedades y son también objetos que pueden tener descendientes. Por ejemplo, si tiene una imagen llamada **Product** en la página, entonces **Product** es un objeto del tipo **Imagen**, pero es también una propiedad del objeto **DOCUMENT** que a su vez es una propiedad del objeto **WINDOW**. La comprensión de la jerarquía, sus objetos y propiedades, es

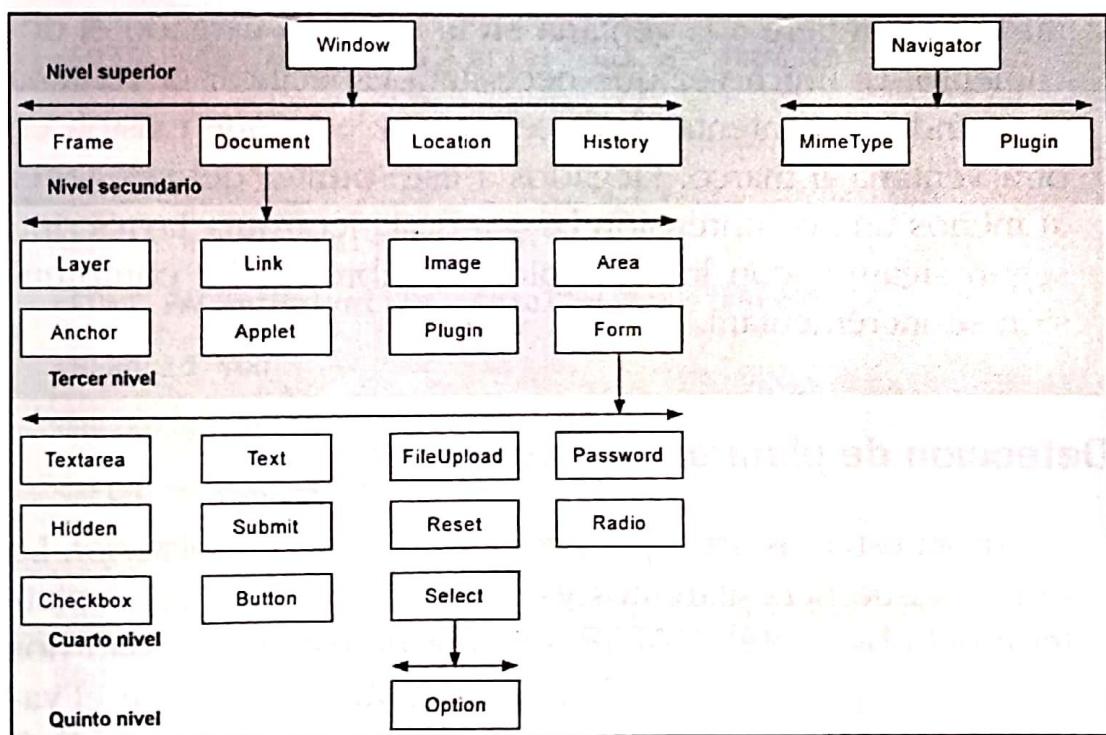


Figura 1.2
La jerarquía JavaScript.

esencial para escribir aplicaciones en JavaScript. Dispone de un glosario de todos los objetos JavaScript y sus propiedades en el Apéndice B, “Objetos de JavaScript”.

Para referenciar un objeto en JavaScript debe llamar al objeto por sí mismo y todos los objetos sobre él en la jerarquía. He aquí un ejemplo de cómo podría referirse al objeto de la imagen, `Product`:

```
document.product
```

Para llamar a una propiedad específica de un objeto siga este mismo modelo y siga un paso más abajo en la jerarquía. La llamada a la propiedad `source` de la imagen `Product` es algo así:

```
document.product.src
```

Notará que el objeto `WINDOW` no estaba incluido como el primer objeto en los ejemplos anteriores. Si no especifica una ventana en particular, JavaScript automáticamente asume que se refiere a la ventana en la que está cargado el documento. La única vez que necesitará especificar la ventana es cuando está intentando acceder a objetos que existen en otra ventana o marco. Llegados a este punto, debería tener al menos una comprensión básica de la jerarquía JavaScript según sigamos con los ejemplos del libro, dicha comprensión se incrementará.

Detección de plataforma/navegador

Ahora estamos listos para comenzar a escribir el script. Lo primero que necesitaremos es acceder a la propiedad `platform` del objeto `NAVIGATOR`, el valor de esta propiedad nos dirá en qué plataforma está el usuario. Vamos a utilizar el valor de esta propiedad como condición en una sentencia `if` para determinar qué tamaño de fuente deseamos en la página.

Una sentencia if es una sentencia condicional que dice que si una condición específica se cumple, sigue adelante y ejecuta el código que sigue; de otra forma, salta a la siguiente sentencia.

En la mayoría de los casos vamos a situar los *scripts* en la parte <HEAD> de la página HTML. Sin embargo, cuando está creando texto dinámicamente necesitará poner al menos parte del *script* en el cuerpo del código HTML donde quiera situar el texto. Para este proyecto situaremos el *script* en el punto donde el texto que deseamos reemplazar dinámicamente esté situado en el documento HTML. He aquí una sección del código HTML que indica donde situaremos el *script*.

```
<TR>
<TD>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</TD>
<TD VALIGN="TOP"><BR><BR>
***** NUESTRO SCRIPT IRÁ AQUÍ *****
<FONT FACE="Helvetica, Arial" COLOR="#000000">
<B>Our Portable
<BR>GenLab is ushering
<BR>in a new age.</B></FONT>
<FONT FACE="Helvetica, Arial" COLOR="#0000FF">
<B>Would you
<BR>&nbsp;&nbsp;&nbsp;&nbsp;like to see it?</B>
</FONT></FONT></TD>
<TD>
<BR>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<IMG
SRC="images/h_read.gif" WIDTH=255 HEIGHT=175 BORDER=0
ALT="Read This"></TD>
</TR>
</TABLE>
```

Ahora que sabemos donde situar el *script* en el código HTML, sigamos adelante e insertemos el primer trozo de *script*.

```
<SCRIPT LANGUAGE="JavaScript">  
  <!--El código tras esto será ignorado por los navegadores  
  → más antiguos  
    // Asigna el tipo de plataforma a una Variable  
    var platform = navigator.platform.substr(0,3);  
    ...  
    // Dejamos de ocultar el código aquí-->  
</SCRIPT>
```

Analicemos esta primera parte de código línea por línea y veamos lo que hace. Primero, insertamos la etiqueta de apertura **<SCRIPT>** como vimos al comienzo del capítulo. Para este *script* sólo vamos a utilizar código soportado por todas las versiones de JavaScript, por lo que no necesitaremos especificar para qué versión estamos codificando con el atributo **LANGUAGE** de la etiqueta **<SCRIPT>**. Después de la etiqueta **<SCRIPT>**, comenzamos la línea de comentario de HTML, que ocultará el código en navegadores que no soporten JavaScript. Estas dos primeras líneas serán el principio de todos los *scripts* que escribamos, así que acostumbrese a verlas.

La tercera línea es un comentario JavaScript; como en HTML, puede añadir anotaciones al código para ayudar a que los *scripts* resulten más fáciles de seguir. Hay dos formas de insertar un comentario en el código JavaScript: la primera (el método que estamos utilizando aquí) es comenzar una línea de código con dos barras.

```
// Esto es tratado como un comentario en JavaScript
```

En este método cualquier código que sigue a las barras en la misma línea se trata como un comentario, y no será interpretará como código de ejecución. El segundo método

utiliza cuando insertamos un comentario que ocupa varias líneas. Para ello, encerraremos el comentario entre estos símbolos, /* y */. Por ejemplo:

```
/* Éste es un ejemplo de un comentario que ocupa múltiples  
líneas de código */
```

Cuando escribamos los *scripts* utilizaremos estos comentarios JavaScript para identificar cada sección principal de código y especificar el propósito básico.

Tras haber insertado el comentario podemos avanzar hacia el cuerpo del *script*. La cuarta línea utiliza el operador básico de asignación (=) para asignar el valor de la propiedad platform a la variable platform.

En esta línea de código ocurren varias cosas nuevas; por tanto, vamos a desglosarla y echemos una mirada más atenta a lo que está pasando. Primero, estamos utilizando un operador de asignación para establecer el valor del operador de la izquierda y para que sea el operando de la derecha. Tenemos también cosas que ocurren a ambos lados del operador. En el lado izquierdo estamos creando una variable por primera vez. Las variables juegan un papel muy importante en la creación de *scripts* de cualquier tipo, y JavaScript no es una excepción. Una variable puede ser concebida como un contenedor con un nombre definido por el usuario en el que se puede almacenar información.

Una vez creada una variable puede simplemente llamar al nombre de la variable para acceder a la información que contiene, o reasignarle nueva información. La sintaxis para crear una nueva variable es la que sigue:

```
var nombrevariable;
```

En el lado izquierdo del operador estamos creando una nueva variable llamada platform. Esta variable ya está lista para contener el valor que se le asigna desde el lado derecho del operador. En el lado derecho, estamos ejecutando

dos acciones: primero, llamamos a la propiedad `platform` del objeto `NAVIGATOR`; esta propiedad contiene un valor de cadena que nos dirá sobre qué plataforma está el usuario: si el usuario está sobre una computadora basada en Windows, el valor de `navigator.platform` será “`Win32`” o “`WinNT`”; si el usuario está sobre una computadora Macintosh, devolverá el valor “`Mac`”.

No estamos realmente interesados en conocer qué versión de Windows está ejecutando el usuario, por lo que sería estupendo si pudiésemos encontrar una forma de recoger el valor devuelto por las computadoras basadas en Windows a los tres primeros caracteres: “`Win`”. Despues asignará un único valor a la variable `platform` sin tener en cuenta qué versión de Windows está ejecutando el usuario. Afortunadamente, hay una forma, es la segunda acción que estamos ejecutando en el lado derecho del operador.

El método `substr()` devolverá un número específico de caracteres de cualquier cadena dada. Dado que sabemos que las tres primeras letras devueltas desde todas las versiones de Windows son las mismas, esto resulta perfecto. La sintaxis del método `substr()` es la siguiente:

```
string.substr(inicio, longitud);
```

siendo `inicio` la posición del carácter con el que deseas comenzar y `longitud` el número de caracteres que deseas que el método evalúe. En la línea anterior, indicamos al método `substr()` que comience en el primer carácter y vaya hasta el tercero. El operador de asignación pondrá “`Win`” para las computadoras basadas en Windows o “`Mac`” para las computadoras Macintosh en la variable `platform`.

Ahora tenemos la información necesaria para insertarla en la próxima parte de código: utilizaremos una sentencia `if`_`else if`_`else` para analizar el valor de `platform`. Echemos un vistazo a la sintaxis de esta sentencia.

```
if (condición1)
{
    sentencias1
}

else if(condición2)
{
    sentencias2
}

else
{
    sentencias3
}
```

En la sintaxis precedente tanto `condición1` como `condición2` representan cualquier expresión que devolverá un valor de `true` o `false`. Si `condición1` es `true`, el motor de JavaScript ejecutará el código contenido en `sentencias1`. Sin embargo, si `condición1` es `false`, el motor de JavaScript saltará las `sentencias1` y pasará a analizar `condición2`. Si `condición2` resulta ser `true`, el motor ejecutará `sentencias2`. Si resulta que tanto `condición1` como `condición2` son `false`, se ejecutarán `sentencias3`. La sentencia `if` es una de las estructuras más comúnmente utilizadas dentro de JavaScript, y veremos muchas de ellas a lo largo del libro. Una vez explicado esto, insertemos la siguiente parte de código que contiene la sentencia `if`.

```
<SCRIPT LANGUAGE="JavaScript">
<!--El código tras esto será ignorado por los navegadores
→ más antiguos

    // Asignamos el tipo de plataforma a una variable
var platform = navigator.platform.substr(0,3);

    // Sentencias if analizando qué plataforma está
    // usando el usuario

    if (platform == "Win")</pre>
```

```
    ...
    ...
}

else if (platform == 'Mac')
{
    ...
}

else
{
    ...
}

// Dejamos de ocultar el código aquí-->
</SCRIPT>
```

En la sentencia `if` del código anterior, le estamos diciendo al motor de JavaScript que compruebe primero el valor de `platform` y si es igual a la cadena “Win”, entonces que ejecute las sentencias que siguen dentro de las llaves. Si el valor de `platform` no es “Win”, que salte a la línea `else if` que compara el valor de `platform` con la cadena “Mac”. Si son iguales, sigue adelante y ejecuta las sentencias siguientes que están entre llaves. Si el valor no coincide con “Win” o “Mac”, entonces la sentencia `if` ejecutará el código después de la línea `else`.

Nótese que estamos utilizando el signo igual doble (`==`) para comparar los valores en las condiciones; así es como hacemos las comparaciones en JavaScript. Si utilizásemos un único signo igual, trataría la sentencia como un operador de asignación, reiniciando por tanto, el valor de `platform`.

Ahora tenemos una estructura básica para el *script*, que detectará qué navegador está siendo utilizado para ver la página. Lo único que queda es insertar el código que deseas-

mos que el navegador ejecute para cada una de las plataformas.

Crear HTML dinámico desde dentro de JavaScript

Como el objetivo es que aparezcan fuentes de distinto tamaño para las distintas plataformas, necesitamos que JavaScript escriba la etiqueta dinámicamente. Hay dos sentencias que puede utilizar para escribir en un documento HTML: `document.write()` y `document.writeln()`.

Ambas escribirán lo que vaya entre los paréntesis; sin embargo, `document.writeln()` insertará un salto de línea tras finalizar la escritura. Normalmente no importa el método que utilice, pero para el *script* utilizaremos `document.writeln()` de forma que si ve el código HTML de la página, una vez que se haya escrito, esté formateado para una fácil lectura.

Puede utilizar esos comandos para imprimir diversos tipos de información. Cualquier cosa escrita entre comillas dentro de los paréntesis se escribirá como una cadena y también puede poner dentro variables e incluso otros comandos de JavaScript que se evaluarán antes de ser escritos. Para nuestro propósito, únicamente imprimiremos una cadena que contiene la etiqueta . Insertemos el método `document.writeln()` dentro de las sentencias `if`.

```
<SCRIPT LANGUAGE="JavaScript">  
<!--El código tras esto será ignorado por los navegadores<br/>→ más antiguos  
  
    // Asignamos el tipo de plataforma a una variable  
var platform = navigator.platform.substr(0,3);  
  
    // Sentencias if analizando qué plataforma está  
    // usando el usuario  
  
if (platform == "Win")  
{
```

```
        document.writeln("<FONT SIZE=\\\"3\\\">");  
    }  
    else if (platform == "Mac")  
    {  
        document.writeln("<FONT SIZE=\\\"2\\\">");  
    }  
    else  
    {  
        document.writeln("<FONT SIZE=\\\"3\\\">");  
    }  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Hecho esto, si el usuario está en una computadora Windows, se imprimirá una etiqueta con el atributo SIZE de valor 3. Si el usuario está en una computadora Macintosh, el atributo será 2. Si no se está utilizando ninguna de dichas plataformas (esto es, el usuario está en una computadora Unix), entonces se imprimirá un tamaño 3 de fuente. En la sentencia `document.writeln`, notará la adición de una barra invertida delante de las comillas que rodean al valor numérico del atributo `FONT SIZE`. Las barras se sitúan para decirle a la sentencia `document.writeln` que realmente escriba las comillas en lugar de tratarlas como la comilla de cierre de la sentencia. Esto se denomina caracteres de escape; puede utilizarlo para que JavaScript trate caracteres reservados tal como lo que representan, en lugar de evaluarlos.

ANÁLISIS DEL *SCRIPT*

Enhorabuena, ya ha terminado con el primer *script*. Ahora tenemos un *script* que distribuirá un contenido diferente a los usuarios en diferentes navegadores. Echemos una última mirada al *script* completo y veamos rápidamente cómo lo llevamos a cabo.

```
<SCRIPT LANGUAGE="JavaScript">

<!--El código tras esto será ignorado por los navegadores
  más antiguos

    // Asignamos el tipo de plataforma a una variable
var platform = navigator.platform.substr(0,3);
    // Sentencias if analizando qué plataforma está
    // usando el usuario

if (platform == "Win")
{
    document.writeln("&lt;FONT SIZE=\\"3\\"&gt;");
}

else if (platform == "Mac")
{
    document.writeln("&lt;FONT SIZE=\\"2\\"&gt;");
}

else
{
    document.writeln("&lt;FONT SIZE=\\"3\\"&gt;");
}

// Dejamos de ocultar el código aquí --&gt;
&lt;/SCRIPT&gt;</pre>
```

He aquí los pasos que seguimos para crear este *script*:

1. Se creó una variable para contener una parte específica del valor del objeto `navigator.platform` utilizando el método `substr()`.
2. Luego se utilizó el valor de la variable `platform` como prueba en una sentencia `if` para elegir las sentencias que queríamos que ejecutase la sentencia `if`.
3. Se uso el método `document.writeln()` para imprimir dinámicamente la etiqueta `` adecuada.

Se han mostrado varios aspectos nuevos de JavaScript en este primer *script*:

- Insertar *scripts* en documentos HTML.
- Ocultar los *scripts* a navegadores que no soportan JavaScript.
- La jerarquía de JavaScript.
- Insertar comentarios en el código JavaScript.
- Crear una variable y utilizar un operador de asignación para darle un valor.
- El uso del método `substr()`.
- La sintaxis y el uso de una sentencia `if...else`.
- Los métodos `document.write` y `document.writeln`.
- Utilizar la barra \ para construir los caracteres de escape reservados.

Proyecto II: cómo imprimir información de *copyright* y la fecha de la última modificación

Ahora que el jefe sabe que puede añadir contenido a una página web dinámicamente, él tiene otros proyectos relacionados con la Web que encargarle: la adición de la información de *copyright* y la fecha en la que una página fue modificada por última vez al final de cada página del sitio (véase la Figura 1.3). La adición de la fecha de la última modificación será especialmente útil en el caso del sitio web de Shelley. Hay muchos usuarios que hacen cambios a las páginas de un sitio web y sería estupendo si hubiese una forma fácil de indicar si un archivo ha sido modificado desde la última vez que se editó. Como la mayoría de los proyectos, incluso aunque el jefe no supiera si podría haberse he-

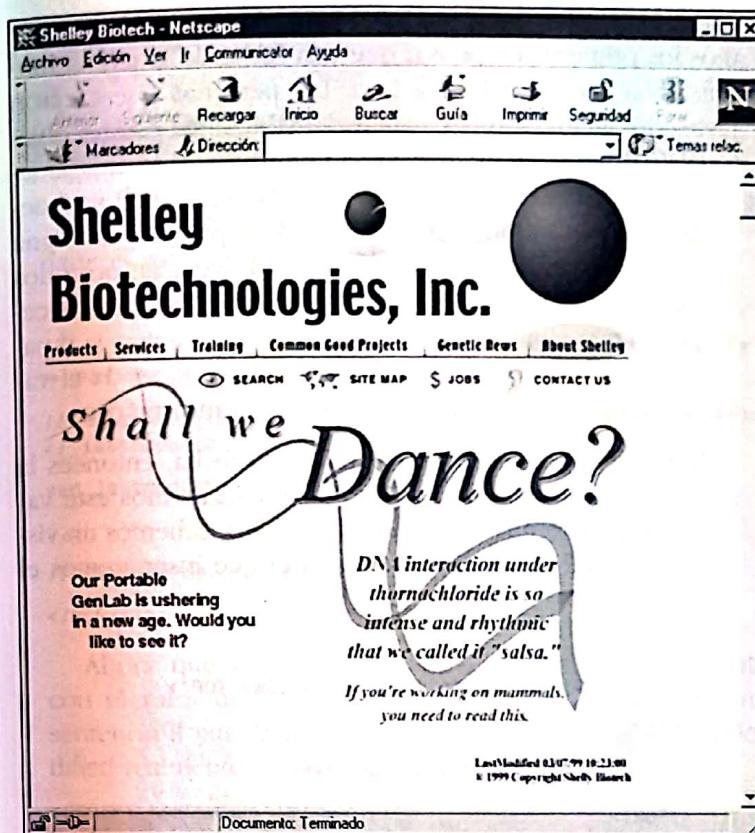


Figura 1.3

Ubicación de la fecha de la última modificación y la información de *copyright*.

cho hace veinte minutos, ahora es de alta prioridad y debe terminarse lo antes posible; así que comencemos.

Iniciación a la propiedad `LastModified`

Después de investigar entre las diferentes propiedades del objeto DOCUMENT, nos encontramos con una propie-

dad que va a hacer el *script* más fácil. Parece ser que no somos los primeros en pensar que una fecha de la última modificación sería una buena idea. Las personas que crearon JavaScript incluyeron la propiedad `lastModified` para el objeto `DOCUMENT`. Cuando se carga una página y se crea la jerarquía JavaScript, el motor va a la cabecera HTTP y obtiene la fecha de la última modificación de la página y la asigna a la propiedad. Sin embargo, hay una desventaja: no todos los servidores web incluyen esta información en la cabecera, por lo que vamos a tener que utilizar una sentencia `if` para asegurarnos de que sólo la imprimiremos si se da el valor. En fin, una sentencia `if` extra nos dará más práctica.

Si el servidor web no proporciona la fecha, entonces la propiedad `lastModified` tendrá valor 0; utilizaremos este valor como prueba en la sentencia `if`. Primero, echemos un vistazo a la parte del código HTML en el que insertaremos el *script*.

```
<TR>
  <TD COLSPAN="3" ALIGN="RIGHT" VALIGN="TOP">
    <FONT SIZE="-2">
      **** NUESTRO SCRIPT IRA AQUI ****
    </FONT>
  </TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Ahora que sabemos donde poner el código, comenzemos insertando las etiquetas `<SCRIPT>` y los comentarios HTML que ocultarán el *script* a los navegadores más antiguos.

```
<SCRIPT LANGUAGE="JavaScript">
```

```
<!--El código tras esto será ignorado por los navegadores
--> más antiguos
...
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

El primer paso del *script* será crear una variable y asignarle el valor de la propiedad `lastModified`.

```
<SCRIPT LANGUAGE="JavaScript">
<!--El código tras esto será ignorado por los navegadores
--> más antiguos
// Asignamos la fecha de última modificación a la variable
// lastmoddate
var lastmoddate = document.lastModified;
...
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Ahora que tenemos una variable llamada `lastmoddate` con el valor de la propiedad `lastModified`, crearemos una sentencia `if` para comprobar y ver que la propiedad `lastModified` realmente contiene la fecha.

```
<SCRIPT LANGUAGE="JavaScript">
<!--El código tras esto será ignorado por los navegadores
--> más antiguos
// Asignamos la fecha de última modificación a la variable
// lastmoddate
var lastmoddate = document.lastModified;
// Creamos una sentencia if para analizar el valor de
// lastmoddate
if(lastmoddate == 0)
{
  ...
}
```

```

else
{
...
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>

```

La sentencia if del código anterior es muy similar a la que utilizamos en el primer *script* de este capítulo; sin embargo, es ligeramente menos complicada dado que su estructura sólo consta de *if...else*. Por tanto, si *lastmoddate* es igual a 0, entonces las sentencias que siguen se ejecutarán y si el valor es distinto de 0, saltará a las sentencias que siguen a *else*.

Ahora que tenemos configurada la sentencia if, echemos un vistazo a lo que queremos imprimir como HTML.

Imprimir dinámicamente datos que no son cadenas de caracteres

En el primer *script* de este capítulo utilizamos el método *document.writeln()* para imprimir una determinada cadena como HTML. Para este *script*, lo ampliaremos para imprimir una combinación de datos tipo cadena y de otros tipos. Primero, sin embargo, utilizaremos el método con el que ya estamos familiarizados para imprimir el contenido necesario si no se encontró ninguna fecha en la propiedad *lastModified*. Si es este el caso, imprimiremos dos líneas:

La primera será una línea indicando que la fecha de la última modificación es desconocida. Luego, insertaremos un salto de línea HTML y la segunda línea, que contiene la información de *copyright*.

```

<SCRIPT LANGUAGE="JavaScript">
<!--El código tras esto será ignorado por los navegadores
más antiguos
</pre>

```

```

// Asignamos la fecha de última modificación a la variable
// lastmoddate
var lastmoddate = document.lastModified;
// Creamos una sentencia if para analizar el valor de
// lastmoddate
if(lastmoddate == 0)
{
document.writeln("Lastmodified: Unknown<BR>&copy; 1999
Copyright Shelley Biotech");
}
else
{
...
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>

```

Hecho esto ahora necesitaremos insertar el código que imprimirá el contenido necesario si se da una fecha en la propiedad *lastModified*. Como se dijo anteriormente, vamos a introducirnos en aspectos ligeramente nuevos en lo que se refiere al método *document.writeln()*. Hasta ahora, hemos hecho que el método imprimiese sólo una cadena; ahora vamos a utilizarlo para escribir una combinación de una cadena de datos y el valor contenido por la variable *lastmoddate*. Esto lo haremos mediante la siguiente línea de código:

```

document.writeln("LastModified: " + lastmoddate +
"<BR>&copy; 1999 Copyright Shelley Biotech")

```

Para combinar los diferentes tipos de datos necesitaremos utilizar el operador de concatenación (+). Este operador tomará dos o más cadenas y las combinará para devolver una nueva. Incluso aunque la variable *lastmoddate* no es una ca-

dene en sí misma, el valor que contiene lo es, por lo que este método funcionará. Por tanto, en la línea anterior estamos combinando primero la cadena `LastModified` con el valor contenido dentro de `lastmoddate`, y luego combinando esa nueva cadena con una cadena que contiene un salto de línea HTML y la información de `copyright`. Veamos cómo queda esta línea de código en el resto del *script*:

```
<SCRIPT LANGUAGE="JavaScript">
<!--El código tras esto será ignorado por los navegadores
  → más antiguos
// Asignamos la fecha de última modificación a la variable
// lastmoddate
var lastmoddate = document.lastModified;
// Creamos una sentencia if para analizar el valor de
// lastmoddate
if(lastmoddate == 0)
{
  document.writeln("Lastmodified: Unknown<BR>&copy; 1999
Copyright Shelley Biotech");
}
else
{
  document.writeln("LastModified: " + lastmoddate +
"<BR>&copy; 1999 Copyright Shelley Biotech");
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

ANÁLISIS DEL SCRIPT

La adición de esa última línea fue la pieza final necesaria para completar este *script*. Ahora tenemos un *script* que imprimirá dinámicamente la fecha de la última modificación y

la información de *copyright* de todas las páginas en las que situamos el *script*. Como con todos los *scripts* del libro, echaremos un vistazo al *script* completado, y qué nuevas áreas cubrimos en el proceso.

```
<SCRIPT LANGUAGE="JavaScript">
<!--El código tras esto será ignorado por los navegadores
  → más antiguos
// Asignamos la fecha de última modificación a la variable
// lastmoddate
var lastmoddate = document.lastModified;
// Creamos una sentencia if para analizar el valor de
// lastmoddate
if(lastmoddate == 0)
{
  document.writeln("Lastmodified: Unknown<BR>&copy; 1999
Copyright Shelley Biotech");
}
else
{
  document.writeln("LastModified: " + lastmoddate +
"<BR>&copy; 1999 Copyright Shelley Biotech");
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

He aquí los pasos que seguimos para crear este *script*:

1. Se asignó el valor de la propiedad `document.lastModified` a la variable `lastmoddate`.
2. Luego fue creada una sentencia `if` para comprobar si el servidor web realmente pasó una fecha de la última modificación al motor de JavaScript.

3. Se insertó una sentencia utilizando el método `document.writeln()` que imprimirá un mensaje indicando que la fecha de la última modificación es desconocida y la información de `copyright`, si no había ninguna fecha contenida en la variable `lastmoddate`.
4. Se escribió una sentencia que utilizaba el método `document.writeln()` para escribir una combinación de datos tipo cadena y la fecha contenida en la variable `lastmoddate`, así como la información de `copyright`, si se daba una fecha de la última modificación.

Veamos los nuevos conceptos que hemos utilizado en este proyecto:

- La propiedad `lastModified` del objeto DOCUMENT.
- Cómo combinar cadenas y valores de distinto tipo con el uso del operador de concatenación (+).

Resumen

Bien, hemos conseguido un buen comienzo con el sitio web de Shelley Biotech. En un período de tiempo relativamente corto hemos añadido dos características muy útiles a la página de inicio. Según vaya pasando el tiempo y aparezcan nuevos navegadores en el mercado que trabajen con HTML de forma diferente, el ser capaz de obtener información del usuario y personalizar las páginas para los visitantes del sitio será cada vez más importante. La otra sección por la que pasamos en este capítulo, la creación dinámica de contenido, tiene un gran potencial para la creación y el mantenimiento eficaz de un sitio web. Según nos adentremos en el libro, veremos muchos ejemplos de lo que se puede hacer utilizando estos conceptos.

Proyectos avanzados

Los *scripts* de este capítulo son ejemplos bastante sencillos de lo que puede realizar con la detección de plataforma/navegador y generación de HTML dinámico.

Pruebe a utilizar estas técnicas en los proyectos siguientes para conseguir una mejor comprensión de los resultados que puede lograr:

1. Utilice la detección de navegador para crear dinámicamente páginas personalizadas para trabajar con versiones de navegadores específicas.
2. Cree una página que recoja toda la información contenida en el objeto NAVIGATOR y genere dinámicamente una tabla que muestre la información al usuario.
(La información obtenida del navegador del usuario y situada en la tabla resulta útil como herramienta de desarrollo.)
3. Si tiene un sitio web donde aparece contenido específico en todas las páginas, utilice el método `document.writeln()` para imprimir el código y guarde el *script* en un archivo JavaScript externo. En cada página, simplemente llame al fichero externo utilizando el atributo SRC de la etiqueta `<SCRIPT>`. Esto le permitirá actualizar sólo un fichero sin afectar demasiado al sitio completo.

Rollovers de imagen



En este capítulo

- Proyecto I: *script del rollover.*
- Proyecto II: cómo añadir funcionalidad avanzada a los *rollovers*.
- Peculiaridades de los *rollovers*.
- Resumen.
- Proyectos avanzados.

Ya ha dado sus primeros pasos con la página de inicio y tiene lo que necesita para hacer el trabajo (ahora su jefe quiere que le sorprenda). Él quiere hacer el sitio más dinámico y darle un poco de agresividad. Ha estado investigando los sitios de sus competidores y todos tienen *rollovers* de imagen (los *rollovers* de JavaScript se utilizan para intercambiar una imagen con una versión diferente de dicha imagen cuando el usuario mueve el cursor por encima de la misma. Y a la inversa, cuando el usuario mueve el cursor fuera de la imagen, la nueva imagen es reemplazada por la original). Esta técnica es ampliamente utilizada en las páginas web, y su director siente que es justo lo que su sitio necesita.

Lo primero es elegir los gráficos que quiere que se vean afectados por los *rollovers*. El mejor lugar para los *rollovers*

de imagen en la mayoría de las páginas son los gráficos principales de navegación, y el sitio de Shelley Biotech no es diferente (véase la Figura 2.1).

Uno de los diseñadores ha llegado con un gran tratamiento gráfico para los *rollovers* y ahora es responsabilidad suya implementarlos (véase la Figura 2.2). Hay tres pasos para crear un *rollover* de imagen: definir los objetos IMAGE, crear

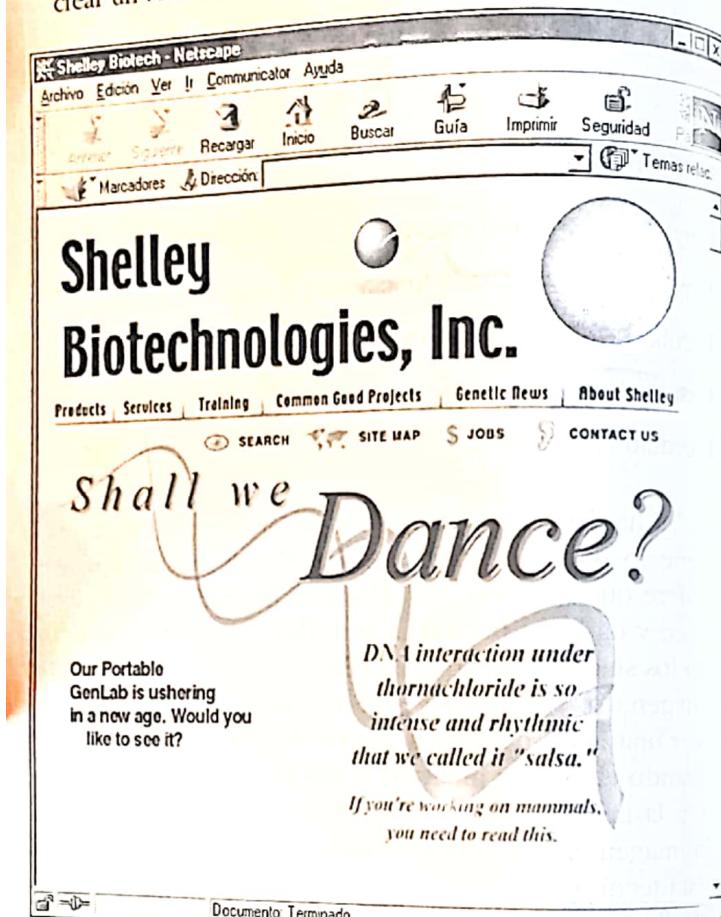


Figura 2.1

Página principal de Shelley Biotech.

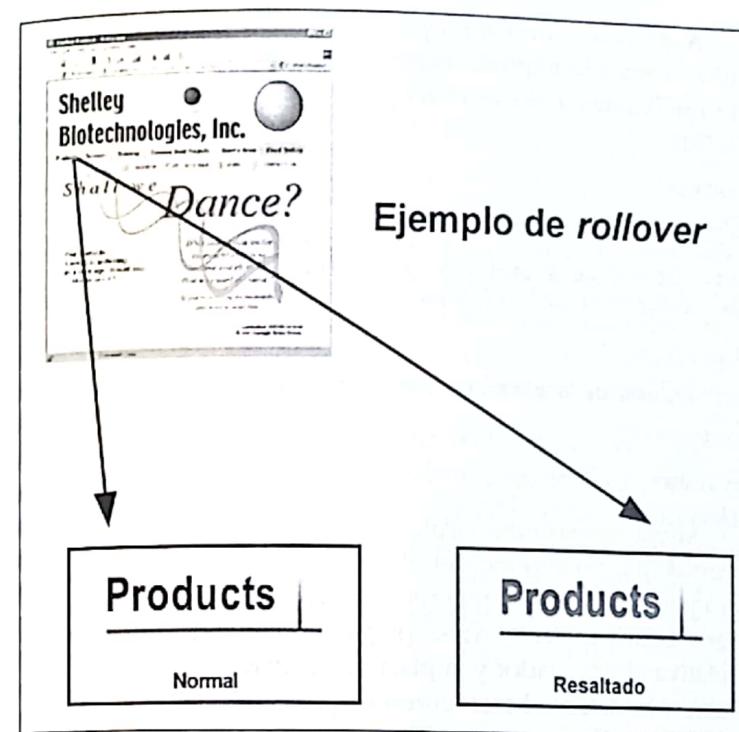


Figura 2.2
Ejemplo de *rollover* de imagen.

la función que hará el trabajo e insertar los manipuladores de eventos de JavaScript necesarios en la imagen y las etiquetas ancla. Primero veremos la creación de los objetos IMAGE.

Proyecto I: script del rollover

Creación de los objetos IMAGE

Antes de que lleguemos a la creación en sí de los objetos IMAGE, debemos poner la etiqueta <SCRIPT> en la página HTML.

A diferencia de los *scripts* del capítulo anterior, no estamos buscando imprimir contenido dentro de la página, por lo que vamos a poner el *script* en la cabecera de la página HTML.

```
<HEAD>
<SCRIPT Language="JavaScript ">
<!-- El código a partir de aquí será ignorado por los
    navegadores más antiguos
...
// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>
```

Ahora necesitamos ocultarlo de los navegadores más antiguos que no soporten el objeto IMAGE, el cual es necesario para el *script* del *rollover*. En el primer capítulo aprendimos cómo utilizar JavaScript para obtener la información relativa al navegador y la plataforma. Podríamos utilizar una variación de aquel método en el que configurábamos varias sentencias if que comprueben todos los navegadores para ver si soportan el objeto IMAGE; sin embargo, esto supondría un montón de trabajo. Afortunadamente, hay una forma más simple. Podemos saber si el navegador del usuario soporta rollovers utilizando una única sentencia if, como se muestra en las líneas de código siguientes:

```
<HEAD>
<SCRIPT Language="JavaScript">
<!-- El Código a partir de aquí será ignorado por
    navegadores más antiguos
    // Creación de los objetos image
if (document.images)
{
...
}
```

```
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>
```

En las líneas de código anteriores utilizábamos document.images como condición de la sentencia if. Devolverá el valor true si el navegador soporta el objeto IMAGE, y el valor false si no lo soporta. Insertando el código dentro de esta sentencia if estamos, en efecto, encubriendo dicho código de los navegadores que no pueden administrar el script.

Cuando se carga una página HTML y el navegador crea los objetos que configuran la jerarquía de JavaScript, cada imagen situada en la página es creada dentro de un objeto y puesta en un array llamado images. Ésta es la primera vez que nos topamos con los arrays, por lo que veamos qué es un array y, en particular, el array images y cómo vamos a utilizarlo.

Puede pensar en un array como en un archivador de carpetas de oficina. Digamos que tiene una página con cuatro imágenes. Cuando el navegador lee el archivo HTML, mueve la página hacia abajo; y cuando llega a la primera imagen, crea un objeto IMAGE para ella y la almacena en el primer cajón del archivador. Cuando llega a la segunda imagen, crea de nuevo un objeto IMAGE; entonces lo pone en el segundo cajón del archivador. Este patrón se mantiene hasta que cada una de las imágenes de la página tiene su propio objeto IMAGE, que se almacena en su propio cajón.

Cuando el navegador necesita hacer referencia a una imagen, mira en el cajón apropiado. También podemos hacer referencia al array y los objetos IMAGE contenidos en él. Hay dos formas de hacer esto. La primera es para hacer referencia a la ubicación en el array donde reside el objeto IMAGE. El problema con este método es que si se añade una imagen

en algún lugar de la página, cambiará la posición de todas las imágenes siguientes. Por tanto, tendrá que recorrer todo el *script* y asegurarse de que ninguna de las imágenes a las que llama se mueve a una posición diferente.

El segundo método, el cual utilizaremos en este *script*, se ocupa de este problema. Si especifica un atributo NAME en las etiquetas , entonces podrá hacer referencia a dicho gráfico en el array utilizando el nombre que le ha asignado. Con este método, incluso si añade otros gráficos a la página, todavía podrá hacer referencia al objeto IMAGE de la misma forma y el *script* no se verá afectado.

Hay seis imágenes de navegación que se verán afectadas por el *script* del *rollover*, por lo que para cada una de dichas imágenes necesitaremos asignar un atributo NAME en las etiquetas .

```
<A HREF="products/index.html"><IMG SRC="images/
h_products_off.gif" WIDTH="71" HEIGHT="33" BORDER="0"
NAME="Products" ALT="Products"></A>
<A HREF="services/index.html"><IMG SRC="images/
h_services_off.gif" WIDTH="67" HEIGHT="33" BORDER="0"
NAME="Services" ALT="Services"></A>
<A HREF="training/index.html"><IMG SRC="images/
h_training_off.gif" WIDTH="75" HEIGHT="33" BORDER="0"
NAME="Training" ALT="Training"></A>
<A HREF="common/index.html"><IMG SRC="images/
h_common_off.gif" WIDTH="157" HEIGHT="33" BORDER="0"
NAME="Common" ALT="Common Good"></A>
<A HREF="genetic/index.html"><IMG SRC="images/
h_news_off.gif" WIDTH="98" HEIGHT="33" BORDER="0"
NAME="News" ALT="Genetic News"></A>
<A HREF="about/index.html"><IMG SRC="images/
h_about_off.gif" WIDTH="106" HEIGHT="33" BORDER="0"
NAME="About" ALT="About Shelley"></A>
```

En las líneas anteriores de HTML hemos añadido un atributo NAME a cada una de las seis etiquetas . El nombre de cada una es la primera palabra de la categoría a la que representa el gráfico. Observe que la primera letra de cada nombre está en mayúscula; JavaScript es sensible a las mayúsculas. Ahora que ya tenemos cuadrados los nombres de dichas imágenes, podemos seguir con la creación de algunos objetos IMAGE nuevos que necesitaremos para el *script*.

Dado que las imágenes que queremos que aparezcan cuando el usuario pase el ratón por encima de uno de los seis gráficos no están situadas explícitamente en la página con código HTML, necesitamos crear dos nuevos objetos IMAGE para cada imagen que queramos intercambiar. Esto añadirá dichas imágenes al array images y nos permitirá acceder a sus propiedades. Comencemos por crear los objetos IMAGE para la imagen Products.

```
<HEAD>
<SCRIPT Language="JavaScript">

</SCRIPT>
</HEAD>
```

En el código precedente estamos inicializando un nuevo objeto IMAGE utilizando un constructor de imagen con la siguiente sintaxis:

`New Image(anchura, altura);`

Poniendo el constructor de imagen al lado derecho del operador de asignación (=), y el nombre que deseamos que tenga el objeto IMAGE a la izquierda, creamos un objeto IMAGE nuevo llamado ProductsOn.

La nomenclatura de estos nuevos objetos es muy importante para el funcionamiento del script. La primera parte del nombre debería ser la misma que el nombre de la imagen correspondiente que ya está en la página. Por ejemplo, en esta primera línea donde estamos creando un objeto para contener la ubicación de la versión intercambiada del gráfico Products, el nombre comienza con Products. Dado que este objeto contiene la versión intercambiada del gráfico, la segunda parte del nombre es On. Cuando se combina, tenemos un nuevo objeto con el nombre ProductsOn. Para cada una de las seis imágenes de navegación vamos a necesitar no sólo un objeto para contener la versión intercambiada del gráfico, sino uno para contener también una versión regular del gráfico. El nombre de este segundo objeto para nuestra imagen Products también comenzará con la palabra "Products", pero terminará en "Off". Añadimos este nuevo objeto IMAGE después del primero.

```
<HEAD>
<SCRIPT Language="JavaScript">
<!-- El código a partir de aquí será ignorado por los
    navegadores más antiguos
    // Creación de objetos image
if (document.images)
{
  ProductsOn=new Image(267, 64);
  ...
  ProductsOff=new Image(267, 64);
  ...
  // Dejamos de ocultar el código aquí --&gt;
&lt;/SCRIPT&gt;
&lt;/HEAD&gt;</pre>

```

```
...
ProductsOff=new Image(267, 64);
...
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>
```

Si se desvía de este convenio de nomenclatura, las funciones que vamos a escribir no sabrán qué objeto IMAGE utilizar para las sustituciones de imágenes y se producirán errores. Ahora tenemos los dos objetos IMAGE nuevos que necesitaremos para el *rollover* Products, pero en este momento están vacíos. Después es necesario asignarles las imágenes reales. Lo hacemos asignando la dirección (URL) de la imagen a la propiedad source del objeto IMAGE, como se muestra a continuación:

```
<HEAD>
<SCRIPT Language="JavaScript" >
<!-- El código a partir de aquí será ignorado por los
    navegadores más antiguos
if (document.images)
{
  ProductsOn=new Image(267, 64);
  ProductsOn.src="images/products_on.gif";
  ProductsOff=new Image(267, 64);
  ProductsOff.src="images/products_off.gif";
  ...
  // Dejamos de ocultar el código aquí --&gt;
&lt;/SCRIPT&gt;
&lt;/HEAD&gt;</pre>

```

Ya hemos terminado los objetos On y Off del rollover products, pero también tenemos que crear objetos para cada una de las otras cinco variantes de la imagen. Lo hacemos utilizando los mismos convenios de nomenclatura que utilizamos para nuestros primeros objetos.

```
<HEAD>
<SCRIPT Language="JavaScript">
<!-- El código a partir de aquí será ignorado por los
    navegadores más antiguos
if (document.images)
{
  ProductsOn=new Image(267, 64);
  ProductsOn.src="images/products_on.gif";
  ProductsOff=new Image(267, 64);
  ProductsOff.src="images/products_off.gif";
  ServicesOn=new Image(267, 64);
  ServicesOn.src="images/services_on.gif";
  ServicesOff=new Image(267, 64);
  ServicesOff.src="images/services_off.gif";
  TrainingOn=new Image(267, 64);
  TrainingOn.src="images/training_on.gif";
  TrainingOff=new Image(267, 64);
  TrainingOff.src="images/training_off.gif";
  CommonOn=new Image(267, 64);
  CommonOn.src="images/common_on.gif";
  CommonOff=new Image(267, 64);
  CommonOff.src="images/common_off.gif";
  GeneticOn=new Image(267, 64);
  GeneticOn.src="images/genetic_on.gif";
  GeneticOff=new Image(267, 64);
  GeneticOff.src="images/genetic_off.gif";
  AboutOn=new Image(267, 64);
  AboutOn.src="images/about_on.gif";
```

```
AboutOff=new Image(267, 64);
AboutOff.src="images/about_off.gif";
...
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>
```

Crear los objetos IMAGE de esta forma tiene otro efecto deseado. Cuando el navegador crea los nuevos objetos IMAGE, carga la imagen en su caché, precargándola consecuentemente en memoria.

Esta carga previa permite que la imagen se muestre inmediatamente sobre el rollover. Es importante tener en cuenta esta precarga cuando cree sus páginas HTML (si no pone los atributos Height, Width y alt en sus etiquetas , la mayoría de los navegadores no mostrarán la página hasta que todo el contenido y las imágenes estén cargados en memoria). Puede desembocar en serios tiempos de carga si tiene muchos objetos IMAGE cargando rollovers en segundo plano, por lo que asegúrese de poner todas las etiquetas necesarias en el código HTML.

Hemos creado con éxito objetos IMAGE para todos los gráficos que necesitamos para llevar a cabo los rollovers. Nuestro próximo paso será crear funciones que ejecuten estos rollovers.

Funciones de rollovers de imagen

Cuando el usuario mueve el ratón sobre uno de estos gráficos, los manipuladores de eventos de JavaScript llaman a la función que vamos a escribir para ejecutar realmente las sustituciones de imágenes. Las funciones son una parte esencial de JavaScript; una función es un conjunto de sen-

tencias JavaScript que ejecutan tareas específicas. Cubriremos tanto la definición como la llamada a funciones en este ejemplo, pero, por ahora, concentrémonos en definir las funciones que necesitamos para los *rollovers*. Se necesitan cuatro elementos básicos para definir una función: la palabra clave **function**, un nombre para la función, un conjunto de argumentos separados por comas y las sentencias que desea que la función ejecute.

Necesitamos crear dos funciones para que los *rollovers* funcionen: una que muestre el gráfico "On" cuando movamos el ratón sobre una imagen y otra que vuelva al gráfico "Off" cuando quitemos el ratón de encima de la imagen. Generalmente, querrá definir las funciones de una página en la cabecera del código HTML, así que comencemos a definir las funciones justo después de los objetos IMAGE.

```
function on(pic)
{
...
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>
```

El primer paso al definir una función es llamar a la palabra clave **function** seguida del nombre de la función; en este caso, la llamaremos **on**. Siguiendo al nombre, dentro de un conjunto de paréntesis, necesitamos poner una lista de variables separadas por comas (una para cada argumento que queramos pasarle a la función). Esas variables nos permitirán pasar valores al interior de la función desde los manipuladores de eventos que llaman a la misma. Sólo necesitamos pasar un único argumento a la función; utilizaremos la variable **pic** para contener dicho argumento.

Luego viene el conjunto de instrucciones que queremos que la función ejecute (estas sentencias están encerradas entre llaves).

```
function on(pic)
{
    if (document.images)
    {
        document.images[pic].src=eval(pic + "On.src");
    }
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>
```

La primera sentencia que queremos que ejecute la función es una sentencia **if** que comprobará si se trata de un navegador antiguo. Si el navegador soporta el objeto IMAGE, se ejecutará la sentencia que sigue a la sentencia **if**. En esta línea le estamos diciendo al navegador que reemplace la imagen original sobre la que está pasando el ratón con la versión **On** del gráfico. Ocurren muchas cosas en esta línea, así que vamos a dividirla en partes más pequeñas. Primero, echemos un vistazo al lado izquierdo del operador de asignación (=).

document.images[pic].src

Estamos haciendo referencia a la propiedad **source** de la imagen situada en la posición del array **images** que tiene el valor de la variable **pic** (digamos que el valor de **pic** es **Products**). Estamos haciendo referencia a la propiedad **source** del objeto IMAGE **Products** y asignándole el valor pasado desde el lado derecho del operador de asignación.

Si hubiéramos llamado explícitamente al gráfico **product** de la siguiente manera:

```
document.product.src
```

necesitaríamos funciones separadas para cada imagen a la que queramos asignarle un *rollover* (no es la forma más eficiente de codificar). Cuando desarrolle sus *scripts*, debería pensar en cómo puede escribir el código de forma más eficiente.

Siempre que pueda utilice las variables de esta forma; le ahorrará mucho tiempo y problemas. Sin embargo, hay ocasiones en las que deseará llamar a una imagen específica cada vez que se llame a la función, y la línea de código anterior es un ejemplo de cómo hacerlo.

En el lado derecho del operador vamos a utilizar la sentencia `eval()` para combinar el valor de `pic` con una cadena para crear un nuevo valor que asignar como origen de la imagen que va a ser intercambiada.

```
eval(pic + "On.src");
```

Si el valor de `pic` es `Products`, la sentencia `eval()` devolverá `ProductsOn.src`, y el valor del origen del objeto IMAGE `ProductsOn` será asignado al origen del objeto IMAGE `Products`. Cuando esto ocurra, la imagen de la página cambiará a la imagen `On` hasta que el usuario retire el ratón de encima de la imagen.

Cuando el usuario retire el ratón de la imagen, se llama a otra función. Ésta es la segunda función que tenemos que crear; la llamaremos `off`.

```
function off(pic)
{
    if (document.images)
    {
        document.images[pic].src= eval(pic + "Off.src");
    }
}
```

```
// Dejamos de ocultar el código aquí ...>
```

```
</SCRIPT>
```

```
</HEAD>
```

Esta función debería ser muy similar a la función `On`; de hecho, la única diferencia es que la segunda mitad de lo que está siendo evaluado en el lado derecho del operador de asignación es `Off.src`, en lugar de `On.src`. Por tanto, cuando esta línea se evalúe, asignará el origen del objeto IMAGE `ProductsOff` para que sea el origen del objeto IMAGE `Products`, devolviendo así la imagen a su apariencia original.

Estas funciones ya están completas y casi hemos terminado con el *script*. Todo lo que nos queda es insertar los manipuladores de eventos JavaScript adecuados en el código HTML.

Inserción de los manipuladores de eventos

Encontrarás que la mayor parte de JavaScript está administrado por eventos. Los eventos son generalmente el resultado de alguna acción por parte del usuario, tal como pasar el ratón sobre un enlace o cambiar el valor de un campo de texto. Para hacer uso de dichos eventos con JavaScript, debe definir un manipulador de eventos para reaccionar a ellos. Hay muchos manipuladores de eventos predefinidos que puedes utilizar para este propósito; comenzaremos utilizando `onMouseOver` y `onMouseOut` para los *rollovers*. Veamos primero cómo queda el código HTML con los manipuladores de eventos.

```
<A HREF="products/index.html"><IMG SRC="images/
h_products_off.gif" WIDTH="71" HEIGHT="33" BORDER="0"
NAME="Products" ALT="Products"> </A>
```

Para nuestros propósitos, debemos añadir los manipuladores de eventos a la etiqueta ANCHOR. Primero, añadimos el manipulador onMouseOver.

```
<A HREF="products.html" onMouseOver="on('products')  
return true;">
```

Cuando se llama a un manipulador de eventos, éste ejecutará cualesquiera sentencias JavaScript que le sigan. Encerramos estas sentencias entre comillas simples y las separamos con punto y coma. Lo primero que queremos que haga nuestro manipulador de eventos es llamar a nuestra función `on`. Para llamar a una función simplemente haga referencia a su nombre y ponga entre comillas simples cualquier valor que desee pasar a la función entre paréntesis separados por comas.

```
on('Products');
```

La sentencia anterior llamará a la función `on` y le pasará el valor `Products`; de nuevo, le estamos pasando `Products` porque ése es el nombre de la imagen a la que queremos que afecte la función. Otra adición común a los *rollovers* es mostrar una frase relevante en la barra de estado del navegador. Para hacer esto necesitamos añadir otra sentencia al manipulador de eventos.

```
<A HREF="products.html" onMouseOver="on('products');  
window.status='Products'; return true;">
```

Llamando a la propiedad de estado del objeto `WINDOW` podemos mostrar cualquier texto en la barra de estado del navegador asignándole un nuevo valor. Lo último que necesitamos añadir al manipulador de eventos es la siguiente sentencia:

```
return true;
```

Esta sentencia le dice al motor de JavaScript que es el fin de lo que necesita ser ejecutado en el manipulador de eventos y que puede seguir con el resto de sus tareas.

Bien, un manipulador de eventos listo, y uno más para finalizar. Añadimos el manipulador `onMouseOut` para que se ocupe de cuando el usuario deja de mover el ratón sobre una imagen.

```
<A HREF="products.html" onMouseOver="on('products');  
window.status='Products'; return true;  
onMouseOut="off('products'); window.status=' ';  
return true;">
```

Para el manipulador `onMouseOut` llamamos a la función `off` y de nuevo le pasamos el nombre del gráfico al que queremos que afecte. Para desactivar la frase que pusimos en la barra de estado, reiniciaremos `window.status` a un valor en blanco y dejaremos el manipulador con la sentencia `return`.

Una vez hayamos insertado los manipuladores de eventos para la imagen `Products`, necesitamos añadir algunos para el resto de las imágenes. La sintaxis será similar (sólo recuerde cambiar el valor que está pasando a las funciones por el nombre del gráfico que deseé modificar).

ANÁLISIS DEL SCRIPT

Ahora deberíamos tener todos los elementos que necesitamos para tener actualizado el sitio web y ejecutándose con algunos buenos *rollovers*. Echemos un vistazo al *script* completo y veamos lo que hicimos en cada sección.

Primero, creamos los objetos `IMAGE` que necesitamos en el *script*.

```
<SCRIPT Language="JavaScript">  
<!-- El código a partir de aquí será ignorado por los  
→ navegadores más antiguos  
if (document.images)  
{
```

```

ProductsOn=new Image(267, 64);
ProductsOn.src="images/products_on.gif";
ProductsOff=new Image(267, 64);
ProductsOff.src="images/products_off.gif";
ServicesOn=new Image(267, 64);
ServicesOn.src="images/services_on.gif";
ServicesOff=new Image(267, 64);
ServicesOff.src="images/services_off.gif";
TrainingOn=new Image(267, 64);
TrainingOn.src="images/training_on.gif";
TrainingOff=new Image(267, 64);
TrainingOff.src="images/training_off.gif";
CommonOn=new Image(267, 64);
CommonOn.src="images/common_on.gif";
CommonOff=new Image(267, 64);
CommonOff.src="images/common_off.gif";
GeneticOn=new Image(267, 64);
GeneticOn.src="images/genetic_on.gif";
GeneticOff=new Image(267, 64);
GeneticOff.src="images/genetic_off.gif";
AboutOn=new Image(267, 64);
AboutOn.src="images/about_on.gif";
AboutOff=new Image(267, 64);
AboutOff.src="images/about_off.gif";
}

```

He aquí los pasos que seguimos para crear los objetos IMAGE necesarios:

1. Configuramos la ocultación a navegadores que no soportan objetos IMAGE.
2. Añadimos el atributo NAME a las seis imágenes sobre las que queremos poner rollovers.

3. Creamos un conjunto de dos nuevos objetos IMAGE para cada una de las seis imágenes que se verán afectadas por el script del *rollover*.
4. Asignamos a la propiedad source de cada objeto IMAGE la ubicación del archivo gráfico adecuado.

Luego, creamos las funciones que ejecutarán los *rollovers*.

```

function on(pic)
{
    if (document.images)
    {
        document.images[pic].src=eval(pic + "On.src");
    }
}

function off(pic)
{
    if(document.images)
    {
        document.images[pic].src= eval(pic + "Off.src");
    }
}

// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>

```

He aquí los pasos que seguimos para crear las funciones del *rollover*:

1. Creamos una función on que cambia la imagen sobre la que está actualmente el usuario por la versión intercambiada de dicho gráfico.

2. Escribimos una función off que devuelve el gráfico a su estado predeterminado cuando el usuario abandona la imagen.
3. Hicimos cambios en las etiquetas <A href> del código HTML existente.

```
<A href="products/index.html"
onMouseOver="on('products'); window.status='Products';
return true; " onMouseOut="off('products');
window.status=' '; return true; ">

<A href="services/index.html"
onMouseOver="on('services'); window.status='Services';
return true; " onMouseOut="off('services');
window.status=' '; return true; ">

<A href="training/index.html"
onMouseOver="on('training'); window.status='Training';
return true; " onMouseOut="off('training');
window.status=' '; return true; ">

<A href="common/index.html" onMouseOver="on('common');
window.status='Common Good Projects'; return true; "
onMouseOut="off('common'); window.status=' '; return true; ">

<A href="genetic/index.html" onMouseOver="on('genetic');
window.status='Genetic'; return true; "
onMouseOut="off('genetic'); window.status=' '; return
true; ">

<A href="about/index.html" onMouseOver="on('about');
window.status='About Us'; return true; "
onMouseOut="off('about'); window.status=' '; return
true; ">
```

He aquí los pasos que seguimos para insertar los manipuladores de eventos en el código HTML:

1. Añadimos los manipuladores del evento onMouseOver a las etiquetas <A href> de cada una de las seis imágenes.

Dentro de dicho manipulador llamamos a la función on, pasándole el nombre del gráfico sobre el que estaba el ratón; también modificamos la barra de estado para mostrar un nuevo mensaje y añadimos la orden return para salir del manipulador.

2. Añadimos los manipuladores del evento onMouseOut para las etiquetas <A href> de cada una de las seis imágenes.

Dentro de cada manipulador llamamos a la función off, pasándole el nombre del gráfico que estábamos intercambiando; reiniciamos la barra de estado y añadimos la orden return para salir del manipulador.

Se introdujeron varios aspectos nuevos de JavaScript en este primer *script*, incluyendo:

- Un método nuevo de ocultación que busca navegadores que soporten los objetos específicos necesarios para el *script*.
- Una introducción a los arrays, en concreto al array images.
- Cómo crear nuevos objetos IMAGE y cómo modificar su propiedad source.
- Cómo crear una función.
- El concepto de manipuladores de eventos y cómo utilizar dos de ellos: onMouseOver y onMouseOut.
- Cómo modificar el mensaje que está siendo mostrado en la barra de estado de la ventana del navegador.

Proyecto II: cómo añadir funcionalidad avanzada a los rollovers

Al jefe le gustan tanto los rollovers de la nueva página de inicio que quiere añadir rollovers a todas las páginas secundarias.

darias del sitio de Shelley (véase la Figura 2.3). Debería un trabajo bastante sencillo considerando que podemos, en la mayor parte, utilizar el mismo *script* que escribimos en la página principal. Sin embargo, tendremos que hacer unos pocos cambios para que funcionen adecuadamente entre las páginas secundarias.

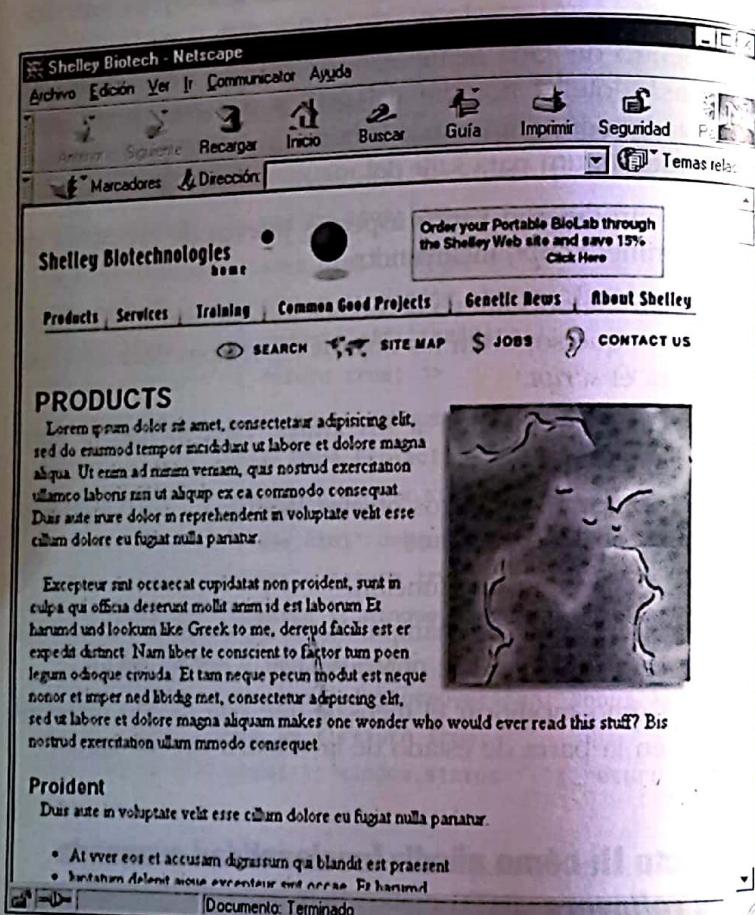


Figura 2.3

Página secundaria de Shelley Biotech.

Lo primero de todo, no queremos que el gráfico de la sección en la que esté actualmente el usuario se vea afectado por un *rollover*; queremos que esté todo el tiempo en su versión On. De esta forma, el usuario siempre podrá saber en qué categoría está. Para hacer esto no tendremos que modificar el *script*; simplemente extraeremos los manipuladores de eventos de la etiqueta <A HREF> del gráfico y modificaremos la imagen a la que se llama en la etiqueta por su versión On.

El otro cambio que necesitamos es un poco más complicado y requiere que modifiquemos las funciones que escribimos para la página principal. Lo que queremos hacer es desactivar el gráfico de la categoría en la que estamos cuando movemos el ratón sobre cualquiera de las otras imágenes de navegación. Esto evitará que dos imágenes estén en la posición On al mismo tiempo, lo que podría resultar confuso para el usuario. También vamos a tener que asegurarnos de que una vez que salgamos de la imagen, el gráfico de la sección en la que estemos vuelva a la posición On.

Inserción de los manipuladores de eventos

Antes de que nos metamos con los cambios necesarios en las funciones, ocupémonos de los cambios en las etiquetas <A HREF> e . Echemos un vistazo a la sección HTML que contiene las imágenes de navegación Products de las páginas secundarias.

```
<A HREF="index.html"><IMG SRC="../images/
products_off.gif" WIDTH="71" HEIGHT="33" BORDER="0"
ALT="Products"></A>
```

```
<A HREF="..../services/index.html"><IMG SRC="../images/
services_off.gif" WIDTH="67" HEIGHT="33" BORDER="0"
ALT="Services"></A>
```

```

<A HREF="../training/index.html"><IMG SRC="../images/
training_off.gif" WIDTH="75" HEIGHT="33" BORDER="0"
ALT="Training"></A>
<A HREF="../common/index.html"><IMG SRC="../images/
common_off.gif" WIDTH="157" HEIGHT="33" BORDER="0"
ALT="Common Good"></A>
<A HREF="../genetic/index.html"><IMG SRC="../images/
news_off.gif" WIDTH="98" HEIGHT="33" BORDER="0"
ALT="Genetic News"></A>
<A HREF="../about/index.html"><IMG SRC="../images/
about_off.gif" WIDTH="106" HEIGHT="33" BORDER="0"
ALT="About Shelley"></A></TD>

```

Primero, necesitamos cambiar la imagen a la que estamos llamando por el gráfico Products en su versión On (`products_on.gif`).

```

<IMG SRC="../images/products_on.gif" WIDTH="71"
HEIGHT="33" BORDER="0" ALT="Products">

```

Luego, cuando el usuario está en la sección Products, la imagen de navegación Products estará constantemente visible. Luego, necesitamos añadir el atributo NAME a las etiquetas `` e insertar los manipuladores de eventos en las etiquetas `<A HREF>` de la imagen. Cuando el usuario llega a esta página, la imagen Products siempre estará visible, por lo que no tenemos que hacer que dicha imagen participe en los *rollovers*. Por tanto, no necesitamos insertar manipuladores de eventos en su etiqueta `<A HREF>`.

```

<A HREF="index.html"><IMG SRC="../images/
products_on.gif" WIDTH="71" HEIGHT="33" BORDER="0"
NAME="Products" ALT="Products"></A>
<A HREF="../services/index.html"
onMouseOver="on('Services'); window.status='Services';
return true;" onMouseOut="off('Services')";
window.status=' ' ; return true;"><IMG SRC="../images/
services_off.gif" WIDTH="67" HEIGHT="33" BORDER="0"
NAME="Services" ALT="Services"></A>

```

```

<A HREF="../training/index.html"
onMouseOver="on('Training'); window.status='Training';
return true;" onMouseOut="off('Training')";
window.status=' ' ; return true;"><IMG SRC="../images/
training_off.gif" WIDTH="75" HEIGHT="33" BORDER="0"
NAME="Training" ALT="Training"></A>
<A HREF="../common/index.html"
onMouseOver="on('Common'); window.status='Common Good
Projects'; return true;" onMouseOut="off('Common')";
window.status=' ' ; return true;"><IMG SRC="../images/
common_off.gif" WIDTH="157" HEIGHT="33" BORDER="0"
NAME="Common" ALT="Common Good"></A>
<A HREF="../genetic/index.html" onMouseOver="on('News')";
window.status='Genetic News'; return true;"'
onMouseOut="off('News'); window.status=' ' ; return
true;"><IMG SRC="../images/news_off.gif" WIDTH="98"
HEIGHT="33" BORDER="0" NAME="News" ALT="Genetic News"></A>
<A HREF="../about/index.html" onMouseOver="on('About')";
window.status='About Us'; return true;"'
onMouseOut="off('About'); window.status=' ' ; return
true;"><IMG SRC="../images/about_off.gif" WIDTH="106"
HEIGHT="33" BORDER="0" NAME="About" ALT="About Shelley">
</A>
</TD>

```

Los manipuladores de eventos deberían resultarnos familiares, porque son los mismos que insertamos en el *script* del *rollover* de la página principal. Encontrar la forma de reutilizar código que ya se está escrito es un buen modo de ahorrar tiempo y mejorar la eficiencia. También se ha añadido el atributo NAME a todas las etiquetas ``; observe que incluso aunque no insertamos manipuladores de eventos para la imagen Products, todavía insertamos el atributo NAME. La razón de esto se aclarará más adelante. Nuestro siguiente paso es crear objetos IMAGE para todos los *rollovers* de las imágenes.

```

<A HREF="../training/index.html"><IMG SRC="../images/
training_off.gif" WIDTH="75" HEIGHT="33" BORDER="0"
ALT="Training"></A>
<A HREF="../common/index.html"><IMG SRC="../images/
common_off.gif" WIDTH="157" HEIGHT="33" BORDER="0"
ALT="Common Good"></A>
<A HREF="../genetic/index.html"><IMG SRC="../images/
news_off.gif" WIDTH="98" HEIGHT="33" BORDER="0"
ALT="Genetic News"></A>
<A HREF="../about/index.html"><IMG SRC="../images/
about_off.gif" WIDTH="106" HEIGHT="33" BORDER="0"
ALT="About Shelley"></A></TD>

```

Primero, necesitamos cambiar la imagen a la que estamos llamando por el gráfico Products en su versión On (`products_on.gif`).

```

<IMG SRC="../images/products_on.gif" WIDTH="71"
HEIGHT="33" BORDER="0" ALT="Products">

```

Luego, cuando el usuario está en la sección Products, la imagen de navegación Products estará constantemente visible. Luego, necesitamos añadir el atributo NAME a las etiquetas `` e insertar los manipuladores de eventos en las etiquetas `<A HREF>` de la imagen. Cuando el usuario llega a esta página, la imagen Products siempre estará visible, por lo que no tenemos que hacer que dicha imagen participe en los rollovers. Por tanto, no necesitamos insertar manipuladores de eventos en su etiqueta `<A HREF>`.

```

<A HREF="index.html"><IMG SRC="../images/
products_on.gif" WIDTH="71" HEIGHT="33" BORDER="0"
NAME="Products" ALT="Products"></A>
<A HREF="../services/index.html"
onMouseOver="on('Services'); window.status='Services';
return true;" onMouseOut="off('Services');
window.status=' ' ; return true;"><IMG SRC="../images/
services_off.gif" WIDTH="67" HEIGHT="33" BORDER="0"
NAME="Services" ALT="Services"></A>

```

```

<A HREF="../training/index.html"
onMouseOver="on('Training'); window.status='Training';
return true;" onMouseOut="off('Training');
window.status=' ' ; return true;"><IMG SRC="../images/
training_off.gif" WIDTH="75" HEIGHT="33" BORDER="0"
NAME="Training" ALT="Training"></A>
<A HREF="../common/index.html"
onMouseOver="on('Common'); window.status='Common Good
Projects'; return true;" onMouseOut="off('Common');
window.status=' ' ; return true;"><IMG SRC="../images/
common_off.gif" WIDTH="157" HEIGHT="33" BORDER="0"
NAME="Common" ALT="Common Good"></A>
<A HREF="../genetic/index.html" onMouseOver="on('News');
window.status='Genetic News'; return true;"'
onMouseOut="off('News'); window.status=' ' ; return
true;"><IMG SRC="../images/news_off.gif" WIDTH="98"
HEIGHT="33" BORDER="0" NAME="News" ALT="Genetic News"></A>
<A HREF="../about/index.html" onMouseOver="on('About')";
window.status='About Us'; return true;"'
onMouseOut="off('About'); window.status=' ' ; return
true;"><IMG SRC="../images/about_off.gif" WIDTH="106"
HEIGHT="33" BORDER="0" NAME="About" ALT="About Shelley">
</A>
</TD>

```

Los manipuladores de eventos deberían resultarnos familiares, porque son los mismos que insertamos en el *script* del *rollover* de la página principal. Encontrar la forma de reutilizar código que ya se está escrito es un buen modo de ahorrar tiempo y mejorar la eficiencia. También se ha añadido el atributo NAME a todas las etiquetas ``; observe que incluso aunque no insertamos manipuladores de eventos para la imagen Products, todavía insertamos el atributo NAME. La razón de esto se aclarará más adelante. Nuestro siguiente paso es crear objetos IMAGE para todos los rollovers de las imágenes.

Creación de los objetos IMAGE

Ésta es otra área en la que tendremos algo de suerte: los objetos IMAGE que creamos para la página principal funcionarán muy bien para todas las páginas secundarias, aunque con ligeras modificaciones. Lo único que tenemos que modificar es la ruta que estamos asignando a la propiedad source del objeto. Dado que las páginas secundarias existen en directorios a un nivel más profundo que la página principal, necesitaremos decirle al objeto que retroceda un nivel antes de mirar en el directorio de las imágenes.

Primero, copiemos los objetos IMAGE existentes de la página principal en la página Products y luego haremos los cambios necesarios.

```
<SCRIPT Language="JavaScript">
<!-- El código a partir de aquí será ignorado por los
    navegadores más antiguos
// Crear Objetos Imagen para los Rollovers
if (document.images)
{
  ProductsOn=new Image(71, 33);
  ProductsOn.src="images/products_on.gif";
  ProductsOff=new Image(71, 33);
  ProductsOff.src="images/products_off.gif";
  ServicesOn=new Image(67, 33);
  ServicesOn.src="images/services_on.gif";
  ServicesOff=new Image(67, 33);
  ServicesOff.src="images/services_off.gif";
  TrainingOn=new Image(75, 33);
  TrainingOn.src="images/training_on.gif";
  TrainingOff=new Image(75, 33);
  TrainingOff.src="images/training_off.gif";
  CommonOn=new Image(157, 33);
  CommonOn.src="images/common_on.gif";
}</pre>

```

```
CommonOff=new Image(157, 33);
CommonOff.src="images/common_off.gif";
NewsOn=new Image(98, 33);
NewsOn.src="images/news_on.gif";
NewsOff=new Image(98, 33);
NewsOff.src="images/news_off.gif";
AboutOn=new Image(106, 33);
AboutOn.src="images/about_on.gif";
AboutOff=new Image(106, 33);
AboutOff.src="images/about_off.gif";
}

...
// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>
```

Ahora necesitamos entrar y modificar la ruta para la propiedad source de los objetos.

```
<SCRIPT Language="JavaScript">
<!-- El código a partir de aquí será ignorado por los
    navegadores más antiguos
// Crear Objetos Imagen para los Rollovers
if (document.images)
{
  ProductsOn=new Image(71, 33);
  ProductsOn.src="../images/products_on.gif";
  ProductsOff=new Image(71, 33);
  ProductsOff.src="../images/products_off.gif";
  ServicesOn=new Image(67, 33);
  ServicesOn.src="../images/services_on.gif";
  ServicesOff=new Image(67, 33);
  ServicesOff.src="../images/services_off.gif";
  TrainingOn=new Image(75, 33);
  TrainingOn.src="../images/training_on.gif";
  TrainingOff=new Image(75, 33);
  TrainingOff.src="../images/training_off.gif";
  CommonOn=new Image(157, 33);
  CommonOn.src="../images/common_on.gif";</pre>

```

```

TrainingOff=new Image(75, 33);
TrainingOff.src="../images/training_off.gif";
CommonOn=new Image(157, 33);
CommonOn.src="../images/common_on.gif";
CommonOff=new Image(157, 33);
CommonOff.src="../images/common_off.gif";
NewsOn=new Image(98, 33);
NewsOn.src="../images/news_on.gif";
NewsOff=new Image(98, 33);
NewsOff.src="../images/news_off.gif";
AboutOn=new Image(106, 33);
AboutOn.src="../images/about_on.gif";
AboutOff=new Image(106, 33);
AboutOff.src="../images/about_off.gif";
}

...
// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>

```

El último paso consiste en insertar las funciones del *rollover* y hacer los cambios necesarios para trabajar sobre las páginas secundarias.

Inserción de las funciones del *rollover*

Podemos comenzar copiando simplemente las funciones que utilizamos en la página principal. Como decíamos anteriormente, necesitaremos modificarlas para trabajar sobre las páginas secundarias; pero dado que el núcleo de la funcionalidad es el mismo, no tendremos que empezar las funciones desde cero.

```

// Funciones del rollover de la imagen
function on(pic)

```

```

(
    if (document.images)
    {
        document.images[pic].src=eval(pic + "On.src");
    }
)
function off(pic)
{
    if(document.images)
    {
        document.images[pic].src= eval(pic + "Off.src");
    }
}

```

Ya tenemos las funciones básicas en la página, así que echemos una mirada más atenta al problema que tenemos que resolver para trabajar adecuadamente. En las páginas secundarias hemos modificado el código HTML de forma que el gráfico de navegación de la sección en la que estamos actualmente está en la posición **On**. Por tanto, cuando el usuario mueve el ratón sobre una de las otras categorías, tanto ese gráfico como el gráfico de la sección en la que se encuentra el usuario estarán resaltados. Esto podría llevar a cierta confusión por parte del usuario, por lo que necesitamos desactivar el gráfico de la sección en la que estamos mientras el usuario mueve el ratón sobre una de las otras categorías.

Para hacer esto vamos a efectuar ligeras modificaciones a las funciones **on** y **off**, así como a crear una nueva función, **off1**, que administrará algo de la funcionalidad adicional que vamos a necesitar. Comencemos con las modificaciones a la función **on**.

```
// Funciones del rollover de la imagen
```

```

...
function on (pic)
{
  if (document.images)
  {
    document.images[pic].src=eval (pic + "On.src");
    document.images['Products'].src=eval
    ("ProductsOff.src");
    ...
  }
}
...

```

La línea que añadimos al *script* le dice al motor de ejecución de JavaScript que desactive el gráfico de la sección Products después de activar el gráfico sobre el que ha pasado el usuario. Observe que hemos llamado específicamente a la imagen Products por el nombre y hemos referenciado explícitamente al objeto origen que queremos asignarle. Por ello, necesitamos modificar esta línea para cada una de las secciones del sitio, de forma que el *script* desactivará la imagen adecuada.

Ahora tenemos la imagen de la sección que estamos desactivando cuando el usuario mueve el ratón sobre otra imagen. Despues, necesitamos una forma de volver atrás cuando el usuario abandona la imagen. La forma más sencilla de llevar a cabo esto es insertar una línea de código en la función off que sólo pone el gráfico de la sección en la que estamos en la posición On cuando se le llama. Desafortunadamente, este método tiene un defecto: si el usuario pasa rápidamente sobre las imágenes de las categorías, una detrás de otra, la imagen de la categoría en la que está el usuario parpadearía al cambiar de una posición a otra. Esto no es el

fin del mundo, pero es un poco enrevesado y requiere una solución más compleja.

Lo que necesitamos es una forma de decir no sólo si el usuario ha dejado de moverse sobre una imagen, sino si el usuario se ha movido o no sobre otra de las imágenes antes de que volvamos a activar la categoría. El primer paso es crear una variable en la que podamos almacenar un valor y utilizarlo para analizar si el usuario ha pasado o no sobre una imagen.

```

// Funciones del rollover de la imagen
var over_checker;
function on (pic)
{
  if (document.images)
  {
    document.images[pic].src=eval (pic + "On.src");
    document.images['Products'].src=eval
    ("ProductsOff.src");
    ...
  }
}
...

```

En el código anterior, declaramos la variable *over_checker* al principio de las funciones del *rollover*. El primer lugar en el que necesitamos utilizar esto es en la función *on*, porque necesitamos saber si el usuario está pasando actualmente sobre la imagen de una categoría. Configuramos *over_checker* para que contenga el valor de *on* cuando se llama a la función *on*.

```

// Funciones del rollover de la imagen
var over_checker;

```

```

function on (pic)
{
  if (document.images)
  {
    document.images[pic].src=eval (pic + "On.src");
    document.images['Products'].src=eval
    ("ProductsOff.src");
    over_checker='on';
  }
}
...

```

No sólo necesitamos saber cuándo el usuario ha pasado sobre una imagen, sino que también necesitamos saber cuándo el usuario ha dejado de pasar sobre una imagen. Para realizar esto necesitamos modificar la función off.

```

function off (pic)
{
  if (document.images)
  {
    over_checker='off';
    document.images[pic].src=eval (pic + "Off.src");
    ...
  }
}
...

```

Igual que en la función on añadimos una línea que asigna un valor a la variable `over_checker`, en este caso estamos asignándole el valor off. Esto nos permitirá saber cuándo el usuario ha dejado de pasar sobre una imagen.

Bien, saber si el usuario está sobre un gráfico está muy bien, pero ¿cómo vamos a utilizar esa información? Entramos en la tercera función. Esta función comprobará la variable `over_checker` y, si no se intercambia nada, entonces pondrá el gráfico de la sección en la que está el usuario de nuevo en la posición On. Echemos un vistazo a la nueva función:

```

function off1 ()
{
  if (over_checker=='off')
  {
    if (document.images)
    {
      document.images['Products'].src=eval
      ("ProductsOn.src");
    }
  }
}

```

Ocurren varias cosas en esta nueva función, pero ningún concepto nuevo que no hayamos cubierto ya. Sin embargo, entremos en ella y veamos qué pasa. Primero, estamos utilizando una sentencia if para comprobar y ver si el valor de `over_checker` es off. Si es el caso, entonces el script baja y comprueba si el navegador del usuario soporta *rollovers* de imagen. Si es así, el script activa entonces el gráfico de la sección en la que está el usuario. Como en la línea de código añadida a nuestra función, requerimos explícitamente que la imagen Products regrese a su estado on. Una vez acabado el código del resto de las secciones secundarias, necesitaremos modificar la imagen afectada.

Hay un trozo final de código que falta en las funciones.

Actualmente, no se llama nunca a la nueva función off1. El lugar lógico para situar la llamada es en la función off, de

forma que cuando el usuario deja de pasar sobre una imagen se llamará a la función off1; y si el usuario no ha pasado sobre otra imagen, entonces el gráfico de la categoría elegida se modificará. Hay sin embargo un problema: si insertamos solamente una llamada normal a la función: dado que el motor de JavaScript interpreta el código tan deprisa, el usuario posiblemente no podría pasar por otra imagen lo suficientemente rápido para evitar que la función active el gráfico de la sección. Por tanto, necesitamos establecer un retraso tal que la función off1 sea llamada tras una breve pausa, permitiendo que el usuario pase sobre una nueva imagen.

Para llevar a cabo esto utilizaremos un nuevo método: setTimeout().

Este método ejecuta sus argumentos después de un período de tiempo definido por el usuario. Echemos un rápido vistazo a su sintaxis.

`setTimeout(argumentos, mseg)`

Para nuestros propósitos, sólo necesitamos darle un argumento (una llamada a la función off1). Sólo se necesita una breve pausa para dar al usuario el tiempo que necesita para pasar sobre una nueva imagen; en este caso, 700 milisegundos servirán. Echemos un vistazo a la nueva línea de código una vez la hayamos añadido a la función off.

```
function off (pic)
{
  if (document.images)
  {
    over_checker='off';
    document.images[pic].src=eval (pic + "Off.src");
    setTimeout ("off1()", 700);
  }
}
```

El último trozo de código que añadimos le dice al motor de ejecución de JavaScript que llame a la función off1 después de 700 milisegundos, que se ocuparán del resto del trabajo por nosotros. Ya terminamos de añadir los *rollovers* avanzados a la sección Products del sitio.

ANÁLISIS DEL SCRIPT

Mientras la mayor parte del *script* para este proyecto se obtuvo a partir de los *rollovers* de la página principal, hicimos algunos cambios y añadimos alguna funcionalidad nueva a los *rollovers*. Revisemos exactamente qué hicimos para este proyecto.

Primero, insertamos los manipuladores de eventos necesarios y realizamos los cambios en las etiquetas .

```
<A HREF="index.html"><IMG SRC="../images/
products_on.gif" WIDTH="71" HEIGHT="33" BORDER="0"
NAME="Products" ALT="Products"></A>

<A HREF="../services/index.html"
onMouseOver="on('Services'); window.status='Services';
return true;" onMouseOut="off('Services');
window.status=' ' ; return true;"><IMG SRC="../images/
services_off.gif" WIDTH="67" HEIGHT="33" BORDER="0"
NAME="Services" ALT="Services"></A>

<A HREF="../training/index.html"
onMouseOver="on('Training'); window.status='Training';
return true;" onMouseOut="off('Training');
window.status=' ' ; return true;"><IMG SRC="../images/
training_off.gif" WIDTH="75" HEIGHT="33" BORDER="0"
NAME="Training" ALT="Training"></A>

<A HREF="../common/index.html"
onMouseOver="on('Common'); window.status='Common Good
Projects'; return true;" onMouseOut="off('Common');
window.status=' ' ; return true;"><IMG SRC="../images/
common_off.gif" WIDTH="157" HEIGHT="33" BORDER="0"
```

```

NAME="Common" ALT="Common Good"></A>
<A HREF=".../genetic/index.html" onMouseOver="on('News');
window.status='Genetic News'; return true;" 
onMouseOut="off('News'); window.status=' ' ; return
true;"><IMG SRC=".../images/news_off.gif" WIDTH="98"
HEIGHT="33" BORDER="0" NAME="News" ALT="Genetic News"></A>
<A HREF=".../about/index.html" onMouseOver="on('About');
window.status='About Us'; return true;" 
onMouseOut="off('About'); window.status=' ' ; return
true;"><IMG SRC=".../images/about_off.gif" WIDTH="106"
HEIGHT="33" BORDER="0" NAME="About" ALT="About Shelley">
</A>
</TD>

```

Echemos un vistazo a los pasos que seguimos en esta parte del *script*:

1. Añadimos los manipuladores del evento `onMouseOver` a las etiquetas `<A HREF>` de todas las imágenes, excepto de la imagen Products.

Dentro de dicho manipulador llamamos a la función `on`, pasándole el nombre del gráfico por el que estábamos pasando; modificamos la barra de estado para mostrar un nuevo mensaje y añadimos la orden `return` para salir del manipulador.

2. Añadimos el manipulador del evento `onMouseOut` a las etiquetas `<A HREF>` de todas las imágenes, excepto la imagen Products.

Dentro de dicho manipulador llamamos a la función `off`, pasándole el nombre del gráfico por el que estábamos pasando; reiniciamos la barra de estado y añadimos la orden `return` para salir del manipulador.

3. Insertamos el atributo `NAME` a las seis etiquetas ``. Incluso aunque no pongamos manipuladores de eventos dentro de la etiqueta `<A HREF>` de la imagen Products,

todavía necesitábamos darle un atributo `NAME`, porque cambiaremos su estado cuando el usuario pase por una de las otras cinco imágenes.

Después de insertar los manipuladores de eventos y los atributos `NAME`, creamos los objetos `IMAGE` que necesitamos en el *script* del *rollover*.

```

<SCRIPT Language="JavaScript">
<!-- El código a partir de aquí será ignorado por los
--> navegadores más antiguos
// Crear Objetos de Imagen para nuestros Rollovers
if (document.images)
{
  ProductsOn=new Image(71, 33);
  ProductsOn.src=".../images/products_on.gif";
  ProductsOff=new Image(71, 33);
  ProductsOff.src=".../images/products_off.gif";
  ServicesOn=new Image(67, 33);
  ServicesOn.src=".../images/services_on.gif";
  ServicesOff=new Image(67, 33);
  ServicesOff.src=".../images/services_off.gif";
  TrainingOn=new Image(75, 33);
  TrainingOn.src=".../images/training_on.gif";
  TrainingOff=new Image(75, 33);
  TrainingOff.src=".../images/training_off.gif";
  CommonOn=new Image(157, 33);
  CommonOn.src=".../images/common_on.gif";
  CommonOff=new Image(157, 33);
  CommonOff.src=".../images/common_off.gif";
  NewsOn=new Image(98, 33);
  NewsOn.src=".../images/news_on.gif";
  NewsOff=new Image(98, 33);
  NewsOff.src=".../images/news_off.gif";
  AboutOn=new Image(106, 33);

```

```

AboutOn.src="../images/about_on.gif";
AboutOff=new Image(106, 33);
AboutOff.src="../images/about_off.gif";
}

```

He aquí los pasos que seguimos para crear los objetos IMAGE:

1. Copiamos los objetos IMAGE que habíamos creado para los rollovers de la página principal.
2. Entonces apuntamos la propiedad source de los objetos hacia la ruta correcta.
3. Insertamos las funciones que potenciaban el script.

```

// Funciones del rollover de la imagen
var over_checker;
function on (pic)
{
    if (document.images)
    {
        Chapter 2 . Image Rollovers 50
        document.images[pic].src=eval (pic + "On.src");
        document.images['Products'].src=eval
        ("ProductsOff.src");
        over_checker='on';
    }
}
function off (pic)
{
    if (document.images)
    {
        over_checker='off';
        document.images[pic].src=eval (pic + "Off.src");
    }
}

```

```

setTimeout ("off1('Products')", 700);
}

function off1 ()
{
    if (over_checker=='off')
    {
        if (document.images)
        {
            document.images['Products'].src=eval
            ("ProductsOn.src");
        }
    }
}

// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>

```

He aquí los pasos que seguimos para crear las funciones:

1. Copiamos las funciones on y off del script del rollover en la página principal, y las usamos como base para las funciones on y off modificadas.

2. Modificamos la función on:

Creamos la variable over_checker al principio de las funciones del rollover.

Dentro de la función on asignamos el valor on a la variable over_checker.

Añadimos una línea para poner la imagen Products en la posición off.

3. Modificamos la función off:

Añadimos una línea que asigna el valor `off` a la variable `over_checker`.

Insertamos el método `setTimeout`, que llamará a la función `off1` tras un retardo de 700 milisegundos.

4. Creamos la función `off1`.

Esta función ejecuta dos pruebas utilizando sentencias `if`: primero, comprueba si el valor de `over_checker` es `off`, y luego comprueba que el navegador del usuario soporta `rollovers`.

Si ambas pruebas devuelven un valor `true`, entonces la función pone la imagen `Products` en la posición `on`.

Con este *script* hemos introducido algunos aspectos nuevos de JavaScript:

- El método `setTimeout`.
- El concepto de reutilización de *scripts* más antiguos para nuevos usos.

Peculiaridades de los rollovers

Los *rollovers* pueden ser un añadido muy útil e interesante para su sitio web; sin embargo, hay algunas peculiaridades que debe tener en cuenta cuando los use. Primero, no todo el mundo podrá verlos (cualquiera que utilice una versión de Netscape anterior a la 3.0 o Internet Explorer 3.0 no podrá verlos). Aunque esto no sea suficiente para disuadirle de utilizar *rollovers*, debería tenerlo en cuenta. No es una buena idea tener navegación que sólo es accesible mediante el uso de un *rollover*; siempre queda bien incluir al menos texto de navegación de respaldo para los que tengan navegadores más antiguos.

La siguiente peculiaridad tiene que ver con Netscape 3.0: cuando se usan imágenes dentro de tablas anidadas, si tiene una tabla que está anidada dentro de otra, y una imagen está ubicada en la tabla anidada, el motor de JavaScript creará dos entradas de array para ese único gráfico. Si la imagen estuviera en una tabla que estuviese anidada dentro de dos tablas, entonces se crearían tres entradas en el array de imágenes. Esto puede traer problemas si está haciendo referencia a sus imágenes basándose en su ubicación en el array `IMAGE` en lugar de por su nombre.

Una anomalía final de la que debería estar pendiente se produce cuando se utilizan archivos gif animados en los *rollovers*. Si está utilizando la misma imagen animada para múltiples *rollovers* y un único objeto `IMAGE`, se encontrará con que la animación sólo funcionará para el primer *rollover*. En todos los siguientes *rollovers* saltará al último fotograma de la animación.

Para solucionar esto, cree objetos `IMAGE` separados para cada instancia donde desee que el gif animado aparezca sobre un *rollover*.

Resumen

Encontrará que los *rollovers* de imágenes son con diferencia lo más comúnmente utilizado de JavaScript hoy en día en los sitios web. Casi todos los sitios que visita los utilizan de una forma u otra, por lo que un buen conocimiento de cómo funcionan y de cuáles son sus limitaciones puede ser una gran ventaja para un programador. Una vez que haya dominado los fundamentos, utilice ese conocimiento para experimentar con diferentes técnicas de *rollovers*. Se sorprenderá de algunos de los creativos e interesantes *rollovers* que puede inventar.

Proyectos avanzados

1. Crear una página de inicio en la que muchos gráficos cambian cuando se pasa sobre uno de los gráficos de la categoría principal.
2. Crear una página en la que una imagen aleatoria reemplaza a una imagen existente cuando se pasa sobre una de las imágenes de la categoría.

JavaScript para navegación



En este capítulo

- Proyecto I: JavaScript y los menús desplegables.
- Proyecto II: cómo utilizar varios menús desplegables para navegar.
- Proyecto III: cómo utilizar JavaScript en una página de inicio de sesión.
- Resumen.
- Proyectos avanzados.

Hasta ahora ha hecho un buen trabajo añadiendo nueva funcionalidad a la página principal de Shelley Biotech. Ahora que el jefe ha visto lo fabuloso que puede ser JavaScript, ha cambiado de opinión. Le encanta el hecho de poder añadir un menú desplegable a la página principal y permitir que el usuario salte con un solo clic a páginas que están situadas en el mismo sitio, pero en niveles inferiores. Por supuesto, como la mayoría de los proyectos, necesitan hacerse con ASAP, por lo que vamos a empezar con otra tarea.

Después de encontrarse con uno de los diseñadores, cree que ha elegido un buen lugar para colocar el menú desplegable en la página principal (véase la Figura 3.1). Ahora só-

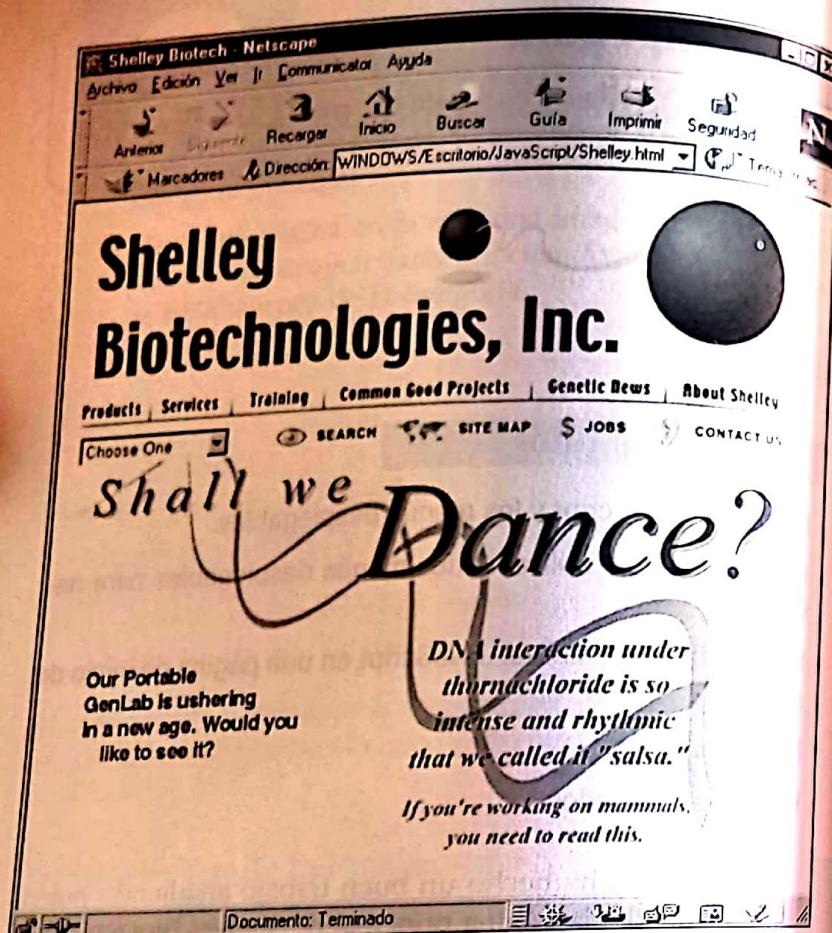


Figura 3.1
Menú desplegable añadido a la página principal.

lo necesitará conseguir los enlaces para dicho menú. Un email al jefe facilitaría la tarea.

Proyecto I: JavaScript y los menús desplegables

Lo primero que necesitamos para crear una página principal es insertar el código HTML que nos facilite el menú

desplegable y el botón Go. Éste es el código que emplearemos:

```
<SELECT NAME="PullDown">
  <OPTION VALUE="default">Get There Quick
  <OPTION VALUE="whatsnew.html">What's New
  <OPTION VALUE="products/featured.html">Feature Product
  <OPTION VALUE="news/press.html">Press Releases
  <OPTION VALUE="store/index.html">Company Store
</SELECT>
<INPUT TYPE="Button" NAME="Go" VALUE="Go">
```

Ya que insertaremos el menú en un diseño preestablecido, vamos a situar las etiquetas `<FORM>` al principio y al final del cuerpo HTML, en lugar de colocarlas a su alrededor para prevenir espacios y saltos de línea no deseados. Observe también que las propiedades de los valores de las opciones tienen espacio en blanco a la izquierda. Los rellenaremos más tarde cuando insertemos el manipulador de evento necesario para ejecutar el *script*.

Creación de la función de navegación

Como el menú desplegable de navegación va a estar en una página que ya tiene *rollovers* de imagen, podemos insertar la función en las etiquetas `<SCRIPT>` existentes. Introduciremos la nueva función después de las funciones de los *rollovers* de imágenes:

```
function off(pic)
{
  if(document.images)
  {
    document.images[pic].src= eval(pic + "Off.src");
  }
}
```

```
}

// Función de navegación del menú desplegable
function PageChanger(page)
{
    ...
}
```

} La función en sí misma será muy simple y consistirá en dos partes. Observe que pasaremos un valor a la función, asignado a la variable `page`. Este valor será la ubicación de la página que el usuario ha seleccionado del menú desplegable. En la primera línea vamos a emplear este valor para cambiar la propiedad de ubicación del objeto `DOCUMENT`. Esta acción volverá a cargar la ventana del navegador con la nueva página web.

```
// Función de navegación del menú desplegable
function PageChanger(page)
{
    document.location= page;
    ...
}
```

Después, añadiremos una línea que servirá como función de almacenamiento. Cuando se ejecute la función, esta línea restablecerá el menú desplegable a la primera elección en dicho menú. En nuestro caso no es muy conveniente, porque la página está cambiando. No obstante, es un toque pequeño muy sutil si el menú desplegable cambia la ubicación de otro marco, en lugar de toda la ventana.

```
// Función de navegación del menú desplegable
function PageChanger(page)
{
    document.location= page;
    document.NavForm.PullDown.options[0].selected=true;
}
```

Inserción del manipulador de eventos

Ahora que ya hemos creado la función, necesitamos insertar el manipulador de eventos en el código HTML para llamar a la función. Antes de hacerlo, vamos a insertar las propiedades de los valores en las opciones del menú desplegable. Como hemos mencionado anteriormente, el valor que hemos pasado a la función es el URL de la página que queremos enviar al usuario. Este valor se toma del atributo VALUE de las opciones del menú. Por tanto, para el valor de cada opción, necesitaremos colocar la localización de la página para cada selección.

```
<SELECT NAME="PullDown">
  <OPTION VALUE="">Get There Quick
  <OPTION VALUE="whatsnew.html">What's New
  <OPTION VALUE="products/featured.html">Feature Product
  <OPTION VALUE="news/press.html">Press Releases
  <OPTION VALUE="store/index.html">Company Store
</SELECT>
```

Para la primera opción del menú, que es sólo un título insertado por motivos de estética, no colocaremos ningún valor, porque no queremos enviar esa opción al usuario. Una vez que el usuario seleccione la opción deseada del menú, necesitará hacer clic en el botón Go para ir a una página nueva. Para lograrlo, insertaremos el manipulador on-Click para el botón Go en el código HTML.

```
<SELECT NAME="PullDown">  
    <OPTION VALUE="">Get There Quick  
    <OPTION VALUE="whatsnew.html">What's New  
    <OPTION VALUE="products/featured.html">Feature Product  
    <OPTION VALUE="news/press.html">Press Releases  
    <OPTION VALUE="store/index.html">Company Store  
</SELECT>
```

```
<INPUT TYPE="Button" NAME="Go" VALUE="Go"
onClick="PageChanger(document.NavForm.PullDown
.options[NavForm.PullDown.selectedIndex].value)">
```

Estamos llevando a cabo dos acciones con el manipulador de eventos: estamos llamando a la función `PageChanger()` y le estamos pasando un valor. El valor que le estamos pasando es el valor de la opción que ha elegido el usuario del menú desplegable. En las funciones creadas anteriormente, el valor que pasamos a la función era una simple cadena. En este manipulador de eventos estamos intentando algo nuevo: estamos haciendo referencia al objeto del menú desplegable y accediendo al valor actualmente seleccionado. Este valor se pasará después a la función y se utilizará para enviar al usuario a la página que desea visitar.

Ahora, el *script* es funcional y está preparado para funcionar. Sin embargo, debido al botón *Go* que hemos añadido a la página, las categorías que están a la derecha del menú desplegable ahora ocupan dos líneas (véase la Figura 3.2). Desgraciadamente esto no es correcto, por lo que deberemos hallar otra solución. Existe otro modo de ejecutar el *script* para que no sólo no tenga este problema, sino que incluso agilice la navegación con el menú desplegable.

Uso de `onChange` para una gratificación instantánea

Para conseguir que la página vuelva a su estado original necesitaremos librarnos del botón *Go*. Sin embargo, por el momento este botón contiene el manipulador de eventos que está ejecutando el *script*. Lo que necesitamos es un manipulador que podamos colocar en el menú desplegable y que ejecute el *script* por nosotros (el manipulador de eventos `onChange`). Este manipulador busca un cambio en el estado del objeto que lo contiene y, cuando lo encuentra, lo lanza.

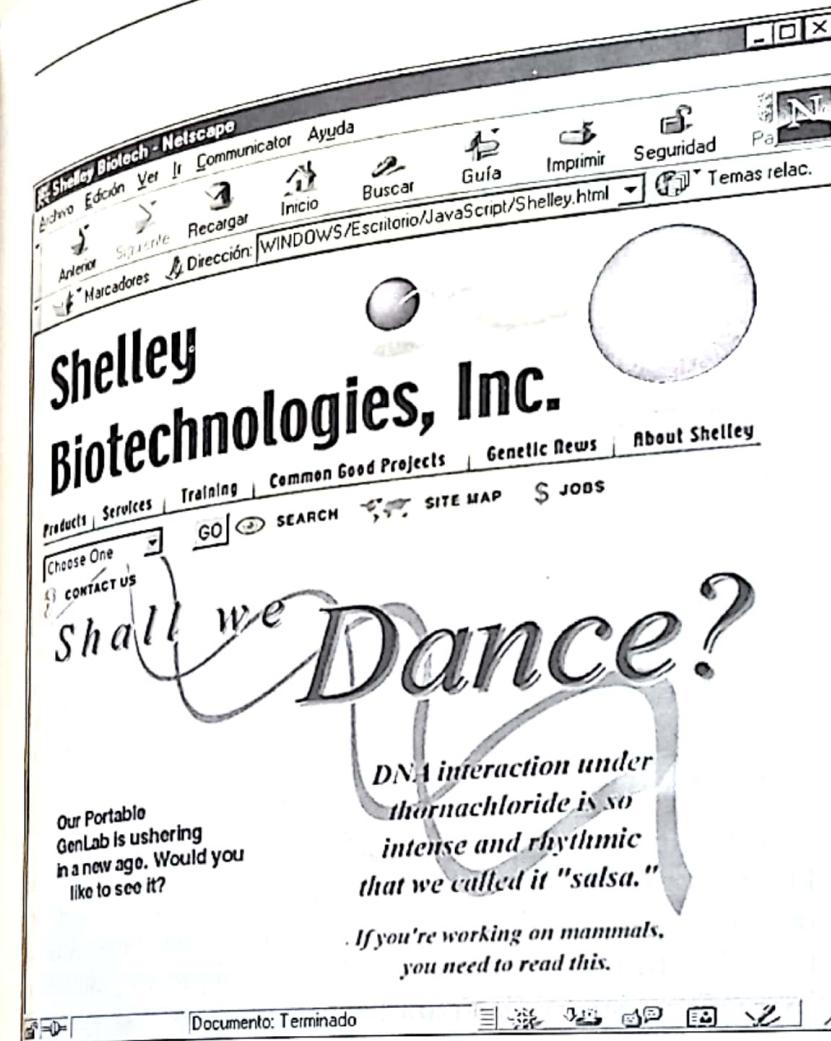


Figura 3.2

Página con un menú desplegable y un botón Go.

Una vez lo hayamos insertado en el menú desplegable, podremos activar la función cuando el usuario cambie la opción del menú (por una distinta a la opción predeterminada que es cargada con la página). Veamos como cambia el código HTML.

```
<SELECT NAME="PullDown"
```

```

onchange="PageChanger(this.options[this.selectedIndex].value)">
<OPTION VALUE=" ">Get There Quick
<OPTION VALUE="whatsnew.html">What's New
<OPTION VALUE="products/featured.html">Feature Product
<OPTION VALUE="news/press.html">Press Releases
<OPTION VALUE="store/index.html">Company Store
</SELECT>

```

Primero, observe que hemos eliminado el botón Go porque ya no era necesario. Después, hemos añadido el manipulador de eventos `onChange` al menú desplegable. El valor que hemos pasado a la función es el mismo (es el valor de la opción que ha seleccionado el usuario). Sin embargo, hemos llamado al valor de un modo diferente al anterior. Como el manipulador de eventos está en el elemento del formulario del que nos gustaría obtener el valor, podemos utilizar el siguiente método:

```
this.options[this.selectedIndex].value
```

Esta línea de código le dice al motor de ejecución de JavaScript que consiga la información del elemento del formulario que contiene el manipulador de eventos. Ambos métodos funcionan igual de bien y siempre es gratificante exponer varias formas para hacer un trabajo.

Al insertar el nuevo manipulador de eventos, el *script* ha finalizado y ya está preparado para trabajar. Como hemos empleado el manipulador `onClick`, ahora la página cambiará cuando el usuario hace una selección en el menú, sin tener que pulsar ningún botón.

ANÁLISIS DEL SCRIPT

Vamos a ver qué es lo que hemos hecho para crear este *script*. Primero creamos una función que acepta un valor de un elemento del formulario, valor que utilizamos para enviar

al usuario a otra página HTML. Necesitamos también dos manipuladores de eventos diferentes que se puedan utilizar para manipular los *scripts*. Primero, veamos la función:

```

// Función de navegación del menú desplegable
function PageChanger(page)
{
    document.location= page;
    document.NavForm.PullDown.options[0].selected=true;
}
</SCRIPT>

```

1. Hemos creado la función PageChanger().

Lo primero que hemos hecho con esta función es ajustar adecuadamente la propiedad de ubicación del objeto DOCUMENT al valor almacenado en la variable `page`. Este valor es el URL que pasa a la función desde el menú desplegable.

Después, hemos restablecido la opción que aparece en el menú desplegable a la primera de las opciones.

Ahora, veamos el código HTML necesario para utilizarlo con el manipulador `onClick`.

```

<SELECT NAME="PullDown">
<OPTION VALUE=" ">Get There Quick
<OPTION VALUE="whatsnew.html">What's New
<OPTION VALUE="products/featured.html">Feature Product
<OPTION VALUE="news/press.html">Press Releases
<OPTION VALUE="store/index.html">Company Store
</SELECT>
<INPUT TYPE="Button" NAME="Go" VALUE="Go"
onClick="PageChanger(document.NavForm.PullDown
.options[NavForm.PullDown.selectedIndex].value)">

```

2. Cuando el usuario haga clic en el botón Go, se llamará a la función. Hemos insertado el manipulador de eventos `onClick` en el elemento de formulario del botón.

En el interior del manipulador de eventos llamamos a la función `PageChanger()`, a la que pasamos el valor que posee la opción de menú seleccionada en el menú desplegable.

Una vez decidido que había que eliminar el botón Go de la página debido a los problemas de diseño, optamos por utilizar el manipulador de eventos `onChange`.

```
<SELECT NAME="PullDown"
onChange="PageChanger(this.options[this.selectedIndex]
.value)">
    <OPTION VALUE="">Get There Quick
    <OPTION VALUE="whatsnew.html">What's New
    <OPTION VALUE="products/featured.html">Feature Product
    <OPTION VALUE="news/press.html">Press Releases
    <OPTION VALUE="store/index.html">Company Store
</SELECT>
```

3. Insertamos el manipulador de eventos `onChange` en el menú desplegable.

De nuevo, en el interior de este manipulador de eventos, llamamos a la función `PageChanger()` y le pasamos el URL de la opción seleccionada en el menú.

Vamos a ver los conceptos nuevos que hemos tenido que abordar durante este proyecto:

- Acceso y cambio de la propiedad de ubicación de un objeto **DOCUMENT**.
- Acceso a los valores de las opciones del menús desplegable y cambio de la opción seleccionada dentro del menú.
- Los manipuladores de eventos `onClick` y `onChange`.

Como los sitios web se han vuelto más y más importantes para el éxito de las empresas, ha aumentado su contenido en saltos y también en limitaciones. Encontrar métodos rápidos y eficientes para navegar a través de la información es hoy en día lo más importante. Al utilizar los manipuladores de eventos `onClick` y `onChange` junto con un menú desplegable, se facilita a los usuarios la información que necesitan.

Proyecto II: cómo utilizar varios menús desplegables para navegar

Con el éxito del *script* de navegación desplegable, el jefe está pensando en otra área de la empresa que se pudiera beneficiar de la navegación mediante un menú desplegable, como la sección de "almacén" de la empresa. Por el momento, para el usuario será engoroso y muy poco intuitivo navegar por tan variado grupo de productos. Shelley Biotech está preparada para lanzar una nueva ola de productos, y sería la oportunidad perfecta para rejuvenecer la sección (véase la Figura 3.3). Sería maravilloso que pudiésemos utilizar los menús desplegables y JavaScript para aumentar la eficiencia y la utilidad.

El jefe quiere dos menús desplegables: uno que contenga una lista con los diferentes grupos de productos, y otro que se pueble dinámicamente con los productos del grupo seleccionado en el primer menú desplegable y envíe al usuario a la página correspondiente al producto elegido.

Esto, en un principio, parece una tarea intimidatoria, pero en los capítulos anteriores hemos aprendido lo que se necesita para realizarla. Dividiremos este proyecto en cuatro secciones: creación de los arrays que mantendrán la información del producto necesaria para cada grupo del producto; creación de la función que poblará dinámicamente el segundo menú; creación de la función que manipulará la

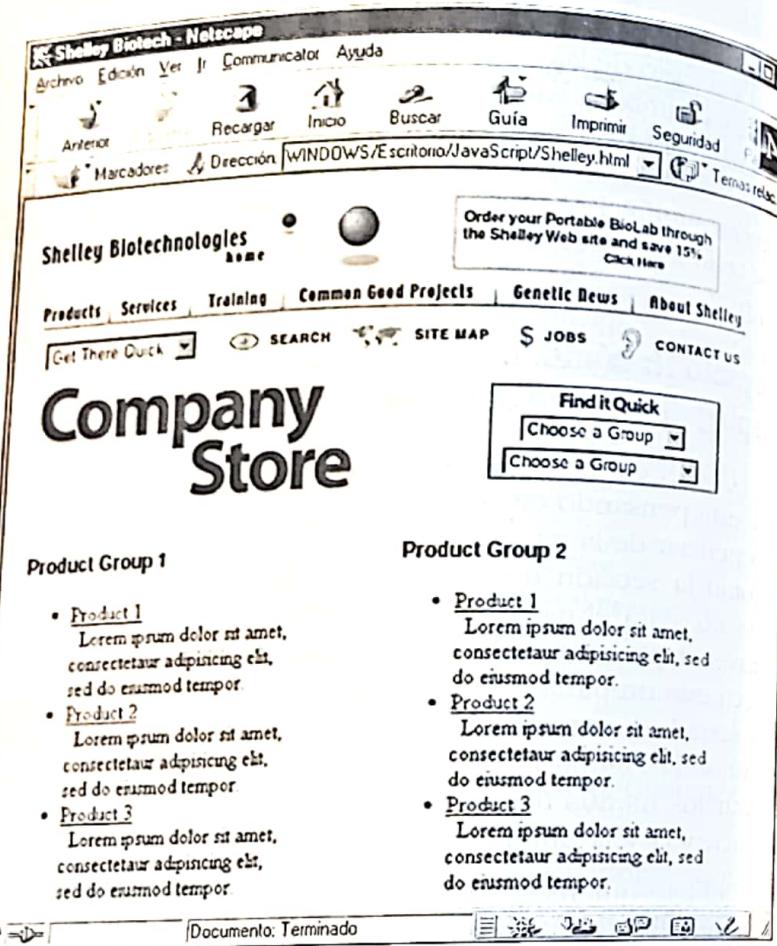


Figura 3.3
La sección Company Store del sitio web Shelley.

navegación; e inserción de los manipuladores de eventos apropiados en el código HTML.

Cómo crear los arrays

Con todos los nuevos productos que están por llegar, el departamento de marketing no está seguro de qué grupos

de productos y qué productos va a incluir en la navegación. Sin embargo, sí saben que por lo menos habrá cuatro grupos de productos que contienen un mínimo de cinco productos cada uno. Utilizando esta información, crearemos un sistema de menús de navegación utilizando marcadores que puedan reemplazarse después por los archivos del producto real. Dicho esto, vamos a comenzar con el trabajo.

Ya hemos realizado una breve introducción del concepto de "array" en los dos primeros capítulos. Hemos aprendido cómo acceder a los arrays creados por el motor de JavaScript mientras se carga una página, y cómo se añaden objetos a algunos de esos arrays. Ahora, sin embargo, necesitaremos crear algunos arrays propios y poblarlos con información actualizada de los productos para completar el trabajo. Para crear un array utilice la siguiente sintaxis:

```
arrayNombreObjeto = new Array(arrayLongitud)
```

En la instrucción anterior, `arrayNombreObjeto` es un marcador para el nombre que va a dar al array, y `arrayLongitud` es el marcador para el número de entradas del array.

Vamos a necesitar dos valores para cada producto y así poder insertar dinámicamente una elección para ellos en el segundo menú desplegable: el primero es el nombre del producto, y el segundo la ubicación de la página HTML a la que corresponde. Para almacenar estos datos crearemos dos arrays para cada grupo de productos: uno para almacenar los nombres de los productos y otro para almacenar las ubicaciones de las páginas. Como no sabemos los nombres de los grupos de productos, usaremos `group1`, `group2`, `group3` y `group4` para identificarlos. Al igual que en el primer proyecto, añadiremos el código nuevo a la etiqueta `<SCRIPT>` existente después del código que ya aparece en la página.

```
// Arrays para los productos
```

```
// Definición de los arrays
```

```

group1_names = new Array(5);
group1_locations = new Array(5);

group2_names = new Array(5);
group2_locations = new Array(5);

group3_names = new Array(5);
group3_locations = new Array(5);

group4_names = new Array(5);
group4_locations = new Array(5);

...

```

Vamos a ver el código que define el primer conjunto de arrays:

```

group1_names = new Array(5);
group1_locations = new Array(5);

```

La primera línea define un nuevo array, `group1_names`, y le asigna una longitud de 5, que es el número de productos que nos han dicho que iban a formar cada grupo (este array almacenará los nombres de los productos de `group1`). La segunda línea define otro nuevo array, `group1_locations`, al que también asigna una longitud de 5 (este array almacenará las ubicaciones de las páginas para los productos de `group1`). Tras definir los arrays para el grupo 1 de productos, hemos creado los arrays para los otros tres grupos.

Ahora que ya hemos definido todos los arrays, necesitamos poblarlos con información sobre los productos. Primero, echemos un breve vistazo a la sintaxis para añadir una entrada al array:

```
arrayNombre[arrayPosición] = "arrayValor"
```

`arrayNombre` es el nombre del array que queremos añadir a la entrada, `arrayPosición` es la posición del array en la que queremos almacenar el valor, y `arrayValor` es el valor que quiere almacenar en el array.

Vamos a poblar los arrays de `group1` con información sobre el producto:

```

// Puebla los arrays de Group1
group1_names[0] = 'Group1_product one';
group1_locations[0] = 'group1/prod_1.html';
group1_names[1] = 'Group1_product two';
group1_locations[1] = 'group1/prod_2.html';
group1_names[2] = 'Group1_product three';
group1_locations[2] = 'group1/prod_3.html';
group1_names[3] = 'Group1_product four';
group1_locations[3] = 'group1/prod_4.html';
group1_names[4] = 'Group1_product five';
group1_locations[4] = 'group1/prod_5.html';
var group1_length = group1_names.length;

```

En el código anterior hemos asignado el nombre de un producto y la ubicación de su página a dos arrays diferentes en la misma posición. En otras palabras, con estas dos líneas:

```

group1_names[0] = 'Group1_product one';
group1_locations[0] = 'group1/prod_1.html';

```

hemos asignado el valor `Group1_product one` a la primera posición del array `group1_names`, y el valor `group1/prod_1.html` a la primera posición del array `group1_locations`. El resto de este código coloca la información de los productos relacionados en los dos arrays y los sitúa de acuerdo con sus números. Al igual que con el nombre de los grupos, vamos a emplear marcadores para los nombres de los productos y las ubicaciones de sus páginas.

Habrá muchas veces en las que esté trabajando en un proyecto y no pueda dar por finalizado el contenido de las páginas. Si emplea el contenido de un marcador, le permitirá terminar el código para que cuando reciba el contenido final, todo lo que tenga que hacer sea pegarlo. El truco está en utilizar valores de marcadores claros y fácilmente describibles, pero sólo en el caso en el que no sea la misma persona que va a insertar el contenido final.

La última línea del código para poblar los arrays de group1 es la siguiente:

```
var group1_length = group1_names.length;
```

Esta línea asigna la longitud de los arrays de group1 a la variable group1_length. Hemos hecho esto para acceder a la longitud del objeto ARRAY. Emplearemos después esta variable en una función que poblará el segundo menú.

Éste es el resto del código que poblará los arrays para los otros grupos de productos. Comparte la misma estructura que en el caso del primer grupo:

```
// Puebla los arrays de Group2
group2_names[0] = 'Group2_product one';
group2_locations[0] = 'group2/prod_1.html';
group2_names[1] = 'Group2_product two';
group2_locations[1] = 'group2/prod_2.html';
group2_names[2] = 'Group2_product three';
group2_locations[2] = 'group2/prod_3.html';
group2_names[3] = 'Group2_product four';
group2_locations[3] = 'group2/prod_4.html';
group2_names[4] = 'Group2_product five';
group2_locations[4] = 'group2/prod_5.html';

var group2_length = group2_names.length;

// Puebla los arrays de Group3
group3_names[0] = 'Group3_product one';
group3_locations[0] = 'group3/prod_1.html';
group3_names[1] = 'Group3_product two';
group3_locations[1] = 'group3/prod_2.html';
group3_names[2] = 'Group3_product three';
group3_locations[2] = 'group3/prod_3.html';
group3_names[3] = 'Group3_product four';
group3_locations[3] = 'group3/prod_4.html';
group3_names[4] = 'Group3_product five';
group3_locations[4] = 'group3/prod_5.html';

var group3_length = group3_names.length;
```

```
// Puebla los arrays de Group4
group4_names[0] = 'Group4_product one';
group4_locations[0] = 'group4/prod_1.html';
group4_names[1] = 'Group4_product two';
group4_locations[1] = 'group4/prod_2.html';
group4_names[2] = 'Group4_product three';
group4_locations[2] = 'group4/prod_3.html';
group4_names[3] = 'Group4_product four';
group4_locations[3] = 'group4/prod_4.html';
group4_names[4] = 'Group4_product five';
group4_locations[4] = 'group4/prod_5.html';

var group4_length = group4_names.length;
```

Ahora tenemos dos arrays para cada grupo de productos poblados con los datos de todos los productos dentro de dichos grupos. Tenemos también cuatro variables que contienen las longitudes de cada uno de los arrays. Con esto ya hecho, pasemos al siguiente punto: crear la función que poblará el segundo menú desplegable.

Creación de la función de población del menú

Antes de comenzar a escribir la función, necesitaremos encontrar un valor dentro de ella. Cuando el usuario seleccione un grupo de productos del primer menú, el script tendrá que eliminar cualquier valor que ocupe el segundo menú antes de insertar los nuevos productos. Esto es muy importante, ya que el nuevo grupo de productos que el usuario selecciona, podría contener algunos productos que ya han sido elegidos previamente. Si se da este caso, los productos del nuevo grupo no podrían sobrescribir los productos del grupo anterior. Para asegurarnos de que eliminamos todos los posibles productos del menú, necesitaremos mediante el siguiente código:

```
// Encontrar la longitud del conjunto de arrays más grande
```

```

var maxLength = group1_length;
if (group2_length > maxLength)
{
    maxLength = group2_length;
}
if (group3_length > maxLength)
{
    maxLength = group3_length;
}
if (group4_length > maxLength)
{
    maxLength = group4_length;
}

```

Este código asigna primero el valor de la longitud del primer conjunto de arrays a la variable `maxLength`. Después emplea las sentencias `if` para comprobar si la longitud de cualquier otro grupo de arrays es más larga que `maxLength` y, por último, asignar el nuevo valor a la variable en tal caso. Al final, `maxLength` almacena el valor del grupo de arrays más largo.

Ahora podemos comenzar con la función. Lo primero que necesitamos hacer es utilizar la variable `maxLength` para eliminar el contenido del segundo menú.

```

// Creación de la función para limpiar el menú
function MenuFiller(choice)
{
    // Limpieza del segundo menú
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        ...
    }
}

```

```

    ++currentPosition;
}
```

```

}
...
)
```

La primera línea del código anterior crea la variable `currentPosition` y le asigna el valor 0. Después tenemos una estructura llamada bucle `while`. Todavía no hemos empleado ninguno, por lo que repasaremos la sintaxis para ver si está en orden.

Ya hemos utilizado las sentencias `if` en los *scripts* (ejecutan una parte de código si una condición específica se cumple). El bucle funciona de modo similar; sin embargo, continúa ejecutando la parte de código que contiene mientras siga cumpliéndose una condición específica. La sintaxis del bucle `while` es la siguiente:

```

while (condición)
{
    sentencias
}
```

Los bucles `while` son muy útiles cuando se quiere cambiar o acceder a muchas propiedades del mismo modo. Sin embargo, los bucles se pueden trucar. Si crea un bucle donde siempre se cumple la condición, se encontrará con algo que se llama bucle “infinito”. En otras palabras, el *script* siempre estará ejecutándose y nunca podrá finalizar, haciendo inútil el *script*, provocando en la mayoría de los casos que el navegador se quede bloqueado.

¿Cómo cambiar la condición que está evaluando para que sea falsa, si no puede salir del bucle hasta que cambie? Estamos encantados de que nos haga esta pregunta. La solución más común es cambiar la condición dentro del bucle.

El bucle while del código anterior efectuará un ciclo mientras la variable currentPosition sea menor que el valor de maxLength. Cada vez que el bucle inicie un ciclo, primero tendremos que eliminar la opción del menú cuya posición corresponda con el valor de currentPosition, y después tendremos que añadir 1 a su valor. De este modo, currentPosition será igual por un momento a maxLength y saldremos del bucle. Ésta también es la primera vez que hemos visto el operador de incremento `++`, que añade 1 al valor de su operando (en este caso, currentPosition). Por tanto, la línea:

```
++currentPosition
```

es la responsable del incremento de la variable.

El bucle efectuará ciclos hasta que el valor de currentPosition sea igual a la longitud del array más largo. De este modo, nos aseguraremos de eliminar todas las posibles opciones del menú. Lograremos la limpieza total del menú colocando el siguiente código en el bucle while:

```
// Creación de la función para limpiar el menú
function MenuFiller(choice)
{
    // Limpieza del segundo menú
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker
            .Products.options[currentPosition].text = ' ';
        document.ProductPicker
            .Products.options[test].value = ' ';
        ++currentPosition;
    }
    ...
}
```

La primera línea del código que hemos insertado hace que el texto de la opción de menú que se corresponden con

el valor actual de la variable currentPosition sea una cadena de espacios en blanco. La segunda línea hace lo mismo con el valor de esta opción de menú. Una vez que el bucle ha ya efectuado todos los ciclos, nos encontraremos con un menú desplegable en blanco (vacío) preparado para aceptar los valores del nuevo grupo de productos.

Ahora que hemos limpiado el menú Products, es hora de comenzar a poblarlo con los productos del nuevo grupo elegido. Antes de conocer los productos que debemos colocar, tenemos que descubrir qué producto ha elegido el usuario. Al igual que en los proyectos anteriores, tendremos al manipulador de eventos pasando un valor a la función que queremos usar para verificar la opción que ha sido elegida. Para esta función, el valor se almacenará en la variable choice. La siguiente parte de código que añadamos será un conjunto de sentencias if para determinar el grupo elegido:

```
// Creación de la función para limpiar el menú
function MenuFiller(choice)
{
    // Limpieza del segundo menú
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker
            .Products.options[currentPosition].text = ' ';
        document.ProductPicker
            .Products.options[test].value = ' ';
        ++currentPosition;
    }

    // Seleccionar el grupo de productos que se ha elegido
    if (choice == 0)
    {
        ...
    }
}
```

```
    else if (choice == 1)
    {
        ...
    }
    else if (choice == 2)
    {
        ...
    }
    else if (choice == 3)
    {
        ...
    }
    else if (choice == 4)
    {
        ...
    }
    ...
}
```

Si observa el código HTML de los menús desplegables, verá que los valores que le hemos dado a los grupos de productos son los números 1 a 4, correspondiendo cada uno de ellos al número de grupo:

```
<SELECT NAME="ProductGroup">
    <OPTION VALUE="0">Choose a Group
    <OPTION VALUE="1">Group One
    <OPTION VALUE="2">Group Two
    <OPTION VALUE="3">Group Three
    <OPTION VALUE="4">Group Four
</SELECT>
<SELECT NAME="Products">
```

Ya que es posible que alguien elija la entrada predeterminada del menú, Choose a Group, hemos asignado un valor 0 a dicha elección. Cuando el manipulador de eventos se pone en funcionamiento, pasa el valor seleccionado a la función, lo asigna a la variable choice y las sentencias if que hemos añadido ejecutarán los comandos que siguen al test que evalúa a true.

Vamos a introducir los comandos que queremos ejecutar después de todas las sentencias if. Estos comandos son necesarios para conseguir dos objetivos: deben asignar a una variable la longitud del array Products del grupo seleccionado y después asignar el nombre del grupo elegido a una variable.

```
// Creación de la función para limpiar el menú
function MenuFiller(choice)
{
    // Limpieza del segundo menú
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker
            .Products.options[currentPosition].text = ' ';
        document.ProductPicker
            .Products.options[test].value = ' ';
        ++currentPosition;
    }
}
```

```

        }

        // Seleccionar el grupo de productos que se ha elegido
        if (choice == 0)
        {
            ...
        }
        else if (choice == 1)
        {
            OutputListSize = group1_length;
            arrayName = "group1";
        }
        else if (choice == 2)
        {
            OutputListSize = group2_length;
            arrayName = "group2";
        }
        else if (choice == 3)
        {
            OutputListSize = group3_length;
            arrayName = "group3";
        }
        else if (choice == 4)
        {
            OutputListSize = group4_length;
            arrayName = "group4";
        }
        ...
    }
}

```

Ya hemos determinado la longitud de los arrays individuales en este *script* anteriormente, y las hemos asignado a variables diferentes, como `group1_length`, `group2_length`, etc. El uso de estos valores más tarde en la función necesitará de una variable sencilla que almacene la longitud del array del grupo elegido. Por tanto, en las líneas de código anteriores, hemos asignado la longitud del array del grupo elegido a la variable `OutputListSize`.

La segunda línea que tenemos que añadir a cada sentencia `if`, asignará el nombre del grupo, si fuese elegido, a la variable `arrayName`. Esta variable se empleará después para ayudar a la información de llamada del array adecuado.

Hasta ahora, hemos cuidado el código para todas las sentencias `if` excepto para la primera, que será `true` si el usuario ha elegido la opción de menú predeterminada, `Choose a Group`. Ya que esta opción no se corresponde con ningún grupo de productos, no hay array `Product` con el que llenar el segundo menú y no tenemos que preocuparnos por encontrar una longitud de array o el nombre de un grupo de productos. Pero hay algo que queremos hacer si se elige esta opción. Ya que el segundo menú se limpia automáticamente cuando se ejecuta la función, necesitaremos volver a poblar la opción predeterminada del menú con el texto `Choose a Group`. Por consiguiente, añadiremos el siguiente comando, que asigna el valor explícitamente a la propiedad `text` de la primera opción del segundo menú. También se ha añadido al *script* un segundo comando, que seleccionará la primera opción del menú.

```

// Creación de la función para limpiar el menú
function MenuFiller(choice)
{
    // Limpieza del segundo menú
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker
            .Products.options[currentPosition].text = ' ';
    }
}

```

Capítulo 3 • JavaScript para navegación

```

document.ProductPicker
.Products.options[test].value = ' ';
++ currentPosition;

}

// Seleccionar el grupo de productos que se ha elegido
if (choice == 0)

{

    document.ProductPicker
    .Products.options[0].text = 'Choose a Group';
    document.ProductPicker
    .Products.options[0].selected=true;

}

else if (choice == 1)

{

    OutputListSize = group1_length;
    arrayName = "group1";

}

else if (choice == 2)

{

    OutputListSize = group2_length;
    arrayName = "group2";

}

else if (choice == 3)

{

    OutputListSize = group3_length;
    arrayName = "group3";

}

else if (choice == 4)

{

    OutputListSize = group4_length;
    arrayName = "group4";
}

```

Ahora conocemos toda la información que se necesita para volver a poblar el segundo menú con los productos del grupo elegido. Vamos a trasladarnos a la sección final de la función, que hará la repoblación. Lo primero que necesitaremos hacer es restablecer el valor de la variable currentPosition a 0, y asegurarnos que la opción predeterminada, Choice a Group, no ha sido seleccionada. Si así fuese, no necesitaremos repoblar el menú, por lo que emplearemos una sentencia if para asegurarnos que choice es mayor que 0.

// Creación de la función para limpiar el menú

```

function MenuFiller(choice)

{
    // Limpieza del segundo menú
    currentPosition = 0;
    while (currentPosition < maxLength)

    {

        document.ProductPicker.Products
        .options[currentPosition].text = ' ';
        document.ProductPicker.Products
        .options[currentPosition].value = ' ';
        ++ currentPosition;

    }

    // Seleccionar el grupo de productos que se ha elegido
    if (choice == 0)

    {

        document.ProductPicker
        .Products.options[0].text = 'Choose a Group';
        document.ProductPicker.Products
        .options[0].selected=true;

    }

    else if (choice == 1)

    {

        OutputListSize = group1_length;
        arrayName = "group1";
    }
}

```

```

    }
    else if (choice == 2)
    {
        OutputListSize = group2_length;
        arrayName = "group2";
    }
    else if (choice == 3)
    {
        OutputListSize = group3_length;
        arrayName = "group3";
    }
    else if (choice == 4)
    {
        OutputListSize = group4_length;
        arrayName = "group4";
    }
    var currentPosition = 0;
    if (choice > 0)
    {
        ...
    }
}

```

Ahora que ya sabemos que se ha elegido un grupo nuevo, podemos comenzar a repoblar el segundo menú. Utilizaremos un bucle while similar al que utilizamos para limpiar el menú al principio de la función. Dicho bucle comparará el valor de currentPosition con la variable OutputListSize, que contiene la longitud de los arrays del producto (Product) elegido.

```

var currentPosition = 0;
if (choice > 0)
{

```

```

while (currentPosition < OutputListSize)
{
    ...
    ++currentPosition;
}
...
}

```

Por supuesto, las sentencias para el bucle serán diferentes. Para repoblar las propiedades text y value de cada opción de menú, necesitaremos colocar dos líneas de código para ejecutar el bucle while.

```

var currentPosition = 0;
if (choice > 0)
{
    while (currentPosition < OutputListSize)
    {
        document.ProductPicker
            .Products.options[currentPosition]
            .text = eval(arrayName +
            '_names[currentPosition]');
        document.ProductPicker
            .Products.options[currentPosition]
            .value = eval(arrayName +
            '_locations[currentPosition]');
        ++currentPosition;
    }
    ...
}

```

Vamos a echar un vistazo detenidamente a las líneas de código anteriores. En la primera mitad de la primera línea, llá-

añadimos a la propiedad `text` de la opción del menú que coincide actualmente con el valor de la variable `currentPosition`.
`document.ProductPicker.Products.options[currentPosition].text`

Cada vez que el bucle `while` efectúa un ciclo, a esta propiedad se le asigna el valor de lo que encuentre en la segunda mitad de la línea.

```
= eval(arrayName + '_names[currentPosition]');
```

Esta parte de la línea es similar a la que empleamos en el Capítulo 2, cuando tratamos los *rollovers* de imágenes. Aquí empleamos el método `eval()` para combinar el valor de la variable `arrayName`, que almacena el nombre del grupo seleccionado, y una cadena, que, cuando se combinan, llaman al array `Product` adecuado y recuperan la entrada en la posición que coincide con el valor actual de `currentPosition`.

Mientras la primera línea se encarga de repoblar la propiedad `text` de la opción en el menú desplegable, la segunda línea repuebla la propiedad `value` de cada opción de menú.

El último comando que debemos añadir a la función asegura de que la primera posición del segundo menú ya poblado, sea la que se elige por defecto.

```
var currentPosition = 0;
if (choice > 0)
{
    while (currentPosition < OutputListSize)
    {
        document.ProductPicker
            .Products.options[currentPosition].text =
        eval(arrayName + '_names[currentPosition]');
        document.ProductPicker
            .Products.options[currentPosition]
            .value = eval(arrayName +
            '_locations[currentPosition]');
```

```
++currentPosition;
}
document.ProductPicker.Products.options[0]
    .selected=true;
}
```

Felicidades, ya hemos terminado con la función. Ésta es, en definitiva, una porción de código avanzado, así que pude estar orgulloso. Sólo dos pasos más, y habremos completado el proyecto. Debemos añadir la función de navegación que llevará al usuario a la página del producto que seleccione y después tenemos que insertar el manipulador de eventos que llamará a la función.

Creación de la función de navegación

Necesitamos una función que mande al usuario a páginas diferentes dependiendo del artículo que elija en el menú desplegable de productos. Un momento, ¿esto no le resulta familiar? Debería. Creamos una función muy parecida al principio de este capítulo. Con unas pequeñas modificaciones podemos utilizar esta función ya existente tanto para una navegación rápida por los enlaces como para el menú desplegable de productos.

Cuando se carga la página y el usuario todavía no ha seleccionado ningún producto, el menú desplegable de productos está vacío, y permanecerá así hasta que se seleccione algún grupo. La adición que tenemos que hacer a la función es una prueba que asegure que cualquier opción que se elija desde el menú desplegable `Products`, será poblada con datos. Si no fuese así, no deseamos que la función de navegación envíe al usuario a una página que no existe. Vamos a echar una mirada a la función que ya tenemos para refrescar la memoria:

```
// Función de navegación del menú desplegable
function PageChanger(page)
{
    document.location= page;
    document.NavForm.PullDown.options[0].selected=true;
}
```

Podemos utilizar el valor de la opción de menú, que se pasa a la función, para hacer la prueba. Introduciremos una sentencia if que indique al motor de JavaScript que ejecute el resto de la función si el valor de la variable page no es igual a (!=) una entrada en blanco.

```
// Función de navegación del menú desplegable
function PageChanger(page)
{
    if (page != ' ')
    {
        document.location= page;
        document.NavForm.PullDown
        .options[0].selected=true;
    }
}
```

Con la adición de una simple sentencia if estamos preparados para modificar la función existente. Recuerde este truco cuando tenga que codificar (no hay motivos para tener que reinventar la rueda todos los días).

Ahora que tenemos a la función enviando al usuario donde desea ir, todo lo que nos queda por hacer es colocar los manipuladores de eventos.

Inserción de los manipuladores de eventos

Necesitaremos insertar dos manipuladores de eventos (uno por cada menú desplegable). El menú desplegable con

los grupos de productos llamará a la función MenuFiller(), mientras que el menú desplegable de productos llamará a la función PageChanger(). Primero veamos el menú desplegable ProductGroup:

```
<SELECT NAME="ProductGroup"
onchange="MenuFiller(this.options[this.selectedIndex]
.value)">
<OPTION value="0">Choose a Group
<OPTION VALUE="1">Group One
<OPTION VALUE="2">Group Two
<OPTION VALUE="3">Group Three
<OPTION VALUE="4">Group Four
</SELECT>
```

El manipulador de eventos resulta familiar. En la mayor parte, es el mismo que el manipulador que utilizamos para el menú desplegable de navegación al principio de este capítulo. La única diferencia es que llamamos a una función diferente. Sin embargo, utilizaremos el mismo manipulador de eventos:

```
<SELECT NAME="Products"
onchange="PageChanger(this.options[this.selectedIndex]
.value)">
<OPTION VALUE="">Choose a
Group &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
<OPTION VALUE="">&nbsp;
<OPTION VALUE="">&nbsp;
<OPTION VALUE="">&nbsp;
<OPTION VALUE="">&nbsp;
</SELECT>
```

Aunque no se lo crea, ya está. Ya hemos acabado. Éste es el script más largo con el que hemos trabajado hasta aho-

ra, así que vamos a revisarlo una última vez como un texto, y ver exactamente qué es lo que hemos logrado.

ANÁLISIS DEL SCRIPT

Lo primero que hemos hecho en este script fue definir el array que almacenará toda la información de los productos de cada grupo de productos:

```
// Arrays para los productos
    // Definición de los arrays
group1_names = new Array(5);
group1_locations = new Array(5);

group2_names = new Array(5);
group2_locations = new Array(5);

group3_names = new Array(5);
group3_locations = new Array(5);

group4_names = new Array(5);
group4_locations = new Array(5);
```

1. Creamos dos arrays para cada grupo de productos; un conjunto de arrays almacenará los nombres de los productos de cada grupo, y el otro almacenará los URL que se corresponden con cada producto.

Una vez creados los arrays, el siguiente paso consiste en poblarlos con la información del producto.

```
// Puebla los arrays de Group1
group1_names[0] = 'Group1_product one';
group1_locations[0] = 'group1/prod_1.html';

group1_names[1] = 'Group1_product two';
group1_locations[1] = 'group1/prod_2.html';

group1_names[2] = 'Group1_product three';
group1_locations[2] = 'group1/prod_3.html';

group1_names[3] = 'Group1_product four';
group1_locations[3] = 'group1/prod_4.html';

group1_names[4] = 'Group1_product five';
group1_locations[4] = 'group1/prod_5.html';
```

```
var group1_length = group1_names.length;
// Puebla los arrays de Group2
group2_names[0] = 'Group2_product one';
group2_locations[0] = 'group2/prod_1.html';

group2_names[1] = 'Group2_product two';
group2_locations[1] = 'group2/prod_2.html';

group2_names[2] = 'Group2_product three';
group2_locations[2] = 'group2/prod_3.html';

group2_names[3] = 'Group2_product four';
group2_locations[3] = 'group2/prod_4.html';

group2_names[4] = 'Group2_product five';
group2_locations[4] = 'group2/prod_5.html';

var group2_length = group2_names.length;
var group3_length = group3_names.length;
// Puebla los arrays de Group3
group3_names[0] = 'Group3_product one';
group3_locations[0] = 'group3/prod_1.html';

group3_names[1] = 'Group3_product two';
group3_locations[1] = 'group3/prod_2.html';

group3_names[2] = 'Group3_product three';
group3_locations[2] = 'group3/prod_3.html';

group3_names[3] = 'Group3_product four';
group3_locations[3] = 'group3/prod_4.html';

group3_names[4] = 'Group3_product five';
group3_locations[4] = 'group3/prod_5.html';

var group3_length = group3_names.length;
var group4_length = group4_names.length;
// Puebla los arrays de Group4
group4_names[0] = 'Group4_product one';
group4_locations[0] = 'group4/prod_1.html';

group4_names[1] = 'Group4_product two';
group4_locations[1] = 'group4/prod_2.html';

group4_names[2] = 'Group4_product three';
group4_locations[2] = 'group4/prod_3.html';

group4_names[3] = 'Group4_product four';
group4_locations[3] = 'group4/prod_4.html';
```

```

group4_names[4] = 'Group4_product five';
group4_locations[4] = 'group4/prod_5.html';
var group4_length = group4_names.length;

2. Con todos los arrays ya poblados, creamos una variable
para cada grupo que almacenará la longitud de cada
array del grupo.

// Encontrar la longitud del conjunto de arrays más grande
var maxLength = group1_length;
if (group2_length > maxLength)
{
    maxLength = group2_length;
}
if (group3_length > maxLength)
{
    maxLength = group3_length;
}
if (group4_length > maxLength)
{
    maxLength = group4_length;
}

3. Una vez que tenemos las longitudes de todos los arrays
almacenadas en las variables, configuramos una serie de
sentencias if para encontrar los grupos más largos y asignar
dicha longitud a la variable maxLength.

```

Después, creamos la función `MenuFiller()`, encargada de poblar los menús desplegables.

```

// Creación de la función para limpiar el menú
function MenuFiller(choice)
{

```

```

// Limpieza del segundo menú
var currentPosition = 0;
while (currentPosition < maxLength)
{
    document.ProductPicker.Products
        .options[currentPosition].text = '';
    document.ProductPicker.Products
        .options[currentPosition].value = '';
    ++currentPosition;
}

// Seleccionar el grupo de productos que se ha elegido
if (choice == 0)
{
    document.ProductPicker.Products.options[0].text =
        'Choose a Group ';
    document.ProductPicker.Products.options[0].value =
        '';
    document.ProductPicker.Products.options[0]
        .selected=true;
}
else if (choice == 1)
{
    OutputListSize = group1_length;
    arrayName = "group1";
}
else if (choice == 2)
{
    OutputListSize = group2_length;
    arrayName = "group2";
}
else if (choice == 3)
{
    OutputListSize = group3_length;
    arrayName = "group3";
}

```

```

    }
    else if (choice == 4)
    {
        OutputListSize = group4_length;
        arrayName = "group4";
    }

    var currentPosition = 0;
    if (choice > 0)
    {
        while (currentPosition < OutputListSize)
        {
            document.ProductPicker
                .Products.options[currentPosition].text =
            eval(arrayName + '_names[currentPosition]');
            document.ProductPicker
                .Products.options[currentPosition]
                .value = eval(arrayName +
                    '_locations[currentPosition]');
            ++currentPosition;
        }
        document.ProductPicker.Products.options[0]
            .selected=true;
    }
}

```

1. Primero, la función limpia cualquier dato del menú desplegable de productos.
2. Despues, la función hace que la variable pase a ella desde el manipulador de eventos y decide qué grupo de productos ha seleccionado el usuario.
3. Finalmente, la función repuebla el menú desplegable de productos con la información del array adecuado.

Con esta función completa, hicimos un ajuste rápido para la función de navegación ya existente.

// Función de navegación del menú desplegable

function PageChanger(page)

{
 if (page != ' ')

{
 document.location= page;
 document.NavForm.PullDown.options[0].selected=true;

}

}

1. Aquí añadimos una sentencia if a la función PageChanger() para asegurar que la opción seleccionada desde el menú desplegable es válida.

Finalmente, añadimos manipuladores de eventos a los dos menús desplegables de productos.

<SELECT NAME="ProductGroup"
onChange="MenuFiller(this.options[this.selectedIndex]
.value)">

<OPTION value="0">Choose a Group

<OPTION VALUE="1">Group One

<OPTION VALUE="2">Group Two

<OPTION VALUE="3">Group Three

<OPTION VALUE="4">Group Four

</SELECT>

<SELECT NAME="Products"

onChange="PageChanger(this.options[this.selectedIndex]
.value)">

<OPTION VALUE=" ">Choose a
Group & & & & &

<OPTION VALUE=" ">&

<OPTION VALUE=" ">&

```
<OPTION VALUE="" >&nbsp;
<OPTION VALUE="" >&nbsp;
<OPTION VALUE="" >&nbsp;
</SELECT>
```

2. Insertamos un manipulador de eventos `onChange` en los menús desplegables.

El manipulador de eventos del menú desplegable `productGroup` llama a la función `MenuFiller()`.

El manipulador de eventos del menú desplegable `products` llama a la función `PageChanger()`.

Vamos a ver qué nuevos conceptos hemos abordado en este proyecto:

- Creación y población de arrays definidos por el usuario.
- El bucle `while`.
- El operador lógico `!=`.
- Acceso a la información y propiedades contenidas en los arrays.
- Acceso y cambio de los valores y propiedades de los elementos de un formulario.

Ahora que ya hemos acabado, hay que llevar rápidamente el trabajo al jefe para que pueda agradecérselo con un premio merecido. Va a convertirse en un destacado guñí de JavaScript. Por supuesto, ahora que ha acabado este proyecto, el jefe ha encontrado otro para que siga trabajando. Vaya, ¡su trabajo es interminable!

Proyecto III: cómo utilizar JavaScript en una página de inicio de sesión

Shelley Biotech está a punto de añadir una nueva sección de socios al sitio. Parte de esta nueva sección no será acce-

sible al público en general. Sólo se permitirá la entrada a los socios existentes. Algunos de los programadores de Perl han creado una página de inicio de sesión (véase la Figura 3.4), escribiendo el *script* correspondiente, a la vez que mantienen una base de datos con los nombres de los usuarios y sus contraseñas. Se preguntará, ¿qué tengo que ver yo con todo esto?

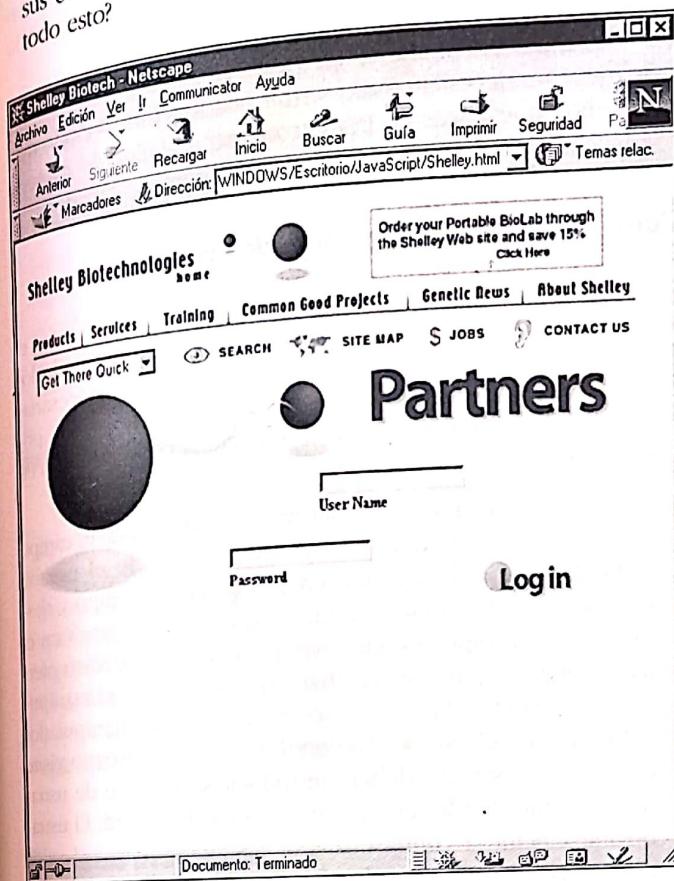


Figura 3.4
Página de inicio de socios.

Pues la creación de una página de inicio de sesión permita al usuario introducir una contraseña y después pulsar Intro para llamar al *script* de conexión. La página de inicio de sesión que los chicos de Perl idearon necesita que el usuario pulse un botón de conexión (Login) para llamar al *script*. El trabajo consiste en tomar la página que han creado y modificarla con JavaScript para que se ejecute cuando se pulse la tecla Intro.

Después del último *script* va a ser cómo pasear por un parque. No necesitaremos escribir una función. Todo lo que tenemos que hacer es insertar el manipulador de eventos adecuado en el código HTML y ya habremos acabado.

Cómo insertar el manipulador de eventos

En los dos últimos *scripts* de este capítulo utilizamos el manipulador de eventos *onChange* en el formulario de los menús desplegables para llamar a las funciones necesarias para ejecutar los *scripts*. Para este *script* utilizaremos *onChange* de nuevo. Veamos cómo funciona con un campo de texto de un formulario.

Cuando el manipulador *onChange* se coloca en el campo de texto de un formulario, se activará cuando el usuario cambia el valor almacenado en el interior del campo y después pulsa Tab o Intro, o bien hace clic con el cursor en el exterior del campo. Esto motiva que el campo de texto pierda el foco, lo que a su vez hace que se ejecute el manipulador de eventos. En este *script* colocaremos el manipulador *onChange* en el campo Password. Cuando un usuario visite esta página, primero deberá introducir su nombre de usuario y después tendrá que pasar al campo Password. El usuario introducirá la contraseña, por lo que cambiará el valor del campo, y después pulsará Intro, o hará clic en el botón Login, lo que motivará que el campo Password pierda el foco y se ejecute el evento. El evento que se lanzará es el

mada de la propiedad **SUBMIT** del formulario que almacena los campos de texto del inicio de sesión.

Vamos a ver cómo queda el código después de insertar el manipulador de eventos:

```
<INPUT TYPE="password" NAME="_User_Pass"
onchange="document.login.submit();">
```

ANÁLISIS DEL SCRIPT

1. Hemos insertado el manipulador de eventos *onChange* en el elemento *password* del formulario.

Eso es todo. Éste es un buen ejemplo de un *script* muy corto que puede ser tremadamente útil. No todos los *scripts* tienen que ser tan largos o tan complejos como el segundo de este capítulo. De hecho, es mejor buscar siempre la manera más eficiente de crear un *script*.

Resumen

Ahora debería conocer mejor cómo emplear JavaScript en la navegación, además de en formularios para mejorar la interactividad y eficiencia. Hemos ampliado los conocimientos sobre los arrays, y aprendido a crearlos y a acceder a la información que tienen almacenada. En el siguiente capítulo abandonaremos el sitio de Shelley Biotechs y comenzaremos a trabajar en el de Stitch Magazine. Tendremos que trabajar con todo lo que hemos aprendido sobre JavaScript y los formularios, a la vez que aprenderemos a manipular errores en JavaScript.

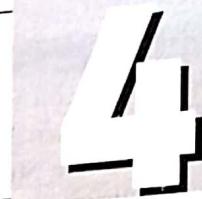
Proyectos avanzados

1. Cree un sistema de tres menús desplegables, donde el segundo menú se pueble basándose en la selección que se

1. haga en el primero, y el tercero se pueble en base a la elección realizada en el segundo.
2. Cree un sistema de tres menús desplegables, donde la elección que se haga en el primero implique la población dinámica de los otros dos.

CAPÍTULO

Manipulación de errores en los formularios



En este capítulo

- Proyecto I: cómo comprobar los campos de formulario vacíos.
- Proyecto II: cómo comprobar datos erróneos en los campos de un formulario.
- Proyecto III: cómo permitir que el usuario sepa qué es erróneo.
- Resumen.
- Proyectos avanzados.

En la segunda mitad de este libro trabajaremos en el sitio web de la revista Stitch (una revista de moda *on-line* ficticia). Es interesante visualmente y sus reporteros son extraordinariamente buenos, siguiendo la pista de diseñadores exclusivos y desenterrando cotilleos embarazosos. Sin embargo, la revista está empezando a crecer y necesita más interactividad para mejorar el contenido de su sitio web. En los dos capítulos siguientes utilizaremos JavaScript para introducir cambios en otros marcos e implementar algo de código JavaScript realmente avanzado para conseguir una experiencia de usuario poco común. En este capítulo aprenderemos a verificar formularios en busca de errores introducidos por el usuario.

El personal de marketing de la revista Stitch está poniendo en marcha una promoción que genere algunas ventas. Para entrar en la tienda, los navegantes deben visitar el sitio web de Stitch e introducir información personal en un formulario (por ejemplo, su nombre, dirección de correo electrónico y lo que quieren saber sobre moda).

Lo cierto es que la mayoría de la gente introduce la información correctamente. Sin embargo, algunos teclearán sus contestaciones, algunos evitarán la información que el departamento de marketing necesita desesperadamente y otros llenarán el formulario con datos basura.

Si incluye algo de código JavaScript para comprobar los errores del formulario antes de que éste se envíe a un script CGI, el personal de marketing tendrá menos datos incompletos o no esenciales que ordenar. La comprobación de errores también hará más fácil la vida del programador CGI, porque puede garantizar que los datos llegarán en el formato correcto. Éste es un *script* bastante largo, de modo que lo dividiremos en tres proyectos: primero, comprobaremos los campos vacíos; después, comprobaremos los datos erróneos introducidos por el usuario, y, finalmente, permitiremos al usuario saber lo que es erróneo. Ya que todos estos proyectos son realmente parte del mismo *script*, revisaremos cada parte al final del capítulo una vez que hayamos completado el *script*.

Proyecto 1: cómo comprobar los campos de formulario vacíos

El personal de marketing quiere que los usuarios rellenen todos los cuadros del formulario, pero sólo una pequeña parte de los datos es realmente necesaria para generar una plantilla (véase la Figura 4.1). Generalmente, debería bastar el nombre y la dirección de correo electrónico como entra-

Figura 4.1

Página del formulario de promoción de Stitch.

das necesarias. Evite pedir un número de teléfono a menos que sea absolutamente necesario (la gente supondrá que es

para tareas de telemarketing y pueden introducir un número que no es el suyo). Aquí está el formulario que el programador HTML de Stitch implementó:

```
<FORM ACTION="/scripts/promotion.asp">
```

METHOD="post">

Fields with a * are required fields.

<P>

<TABLE BORDER="0">

<TR>

<TD>*</TD>

<TD>First Name</TD>

<TD><INPUT TYPE="text" NAME="firstName"></TD>

</TR>

<TR>

<TD>*</TD>

<TD>Last Name</TD>

<TD><INPUT TYPE="text" NAME="lastName"></TD>

</TR>

</TABLE>

<TABLE BORDER="0">

<TR>

<TD COLSPAN="3">

Mailing Address

</TD>

</TR>

<TR>

<TD> </TD>

<TD>Street</TD>

<TD><INPUT TYPE="text" SIZE="40" NAME="address"></TD>

```

<TR>
<TD>&ampnbsp</TD>
<TD><B>City</B></TD>
<TD><INPUT TYPE="text" NAME="address">&ampnbsp&ampnbsp;
<B>State</B><INPUT TYPE="text" NAME="address"
MAXLENGTH="2" SIZE="2">&ampnbsp&ampnbsp;<B>Zip
Code</B><INPUT TYPE="text" NAME="zipCode"
MAXLENGTH="5" SIZE="5">
</TD>
</TR>
</TABLE>
<TABLE>
<TR>
<TD><B><FONT COLOR="#FF0000">*</FONT></B></TD>
<TD><B>Email Address</B></TD>
<TD><INPUT TYPE="text" NAME="email"></TD>
</TR>
<TR>
<TD>&ampnbsp</TD>
<TD><B>Phone Number</B></TD>
<TD>
<INPUT TYPE="text" NAME="phonenumber" SIZE="12"
MAXLENGTH="12">
</TD>
</TR>
<TR>
<TD>&ampnbsp</TD>
<TD><B>Fax Number</B></TD>
<TD>
<INPUT TYPE="text" NAME="faxnumber" SIZE="123"
MAXLENGTH="12">

```

```

</TD>
</TR>
<TR>
    <TD>&nbsp;</TD>
    <TD><B>What interests you most<BR>
        about clothes and fashion?</B>
    </TD>
    <TD>
        <TEXTAREA ROWS="6" COLS="40">Be
        honest</TEXTAREA>
    </TD>
</TR>
<TR>
    <TD><B><FONT COLOR="#FF0000">*</FONT></B></TD>
    <TD>
        <B>What do you want to see in an<BR>
            online fashion magazine like Stitch?</B>
    </TD>
    <TD>
        <SELECT NAME="whatTheyWant">
            <OPTION VALUE="default">Please choose a category
            <OPTION VALUE="trends">Upcoming trends
            <OPTION VALUE="models">Information about models
            <OPTION VALUE="designers">Information about
                designers
            <OPTION VALUE="rage">What the current rage is
            <OPTION VALUE="gossip">Gossip!
        </SELECT>
    </TD>
</TR>
<TR>
    <TD COLSPAN="2"Want to join?</TD>

```

```

<TD><INPUT TYPE="checkbox" NAME="join" SELECTED></TD>
</TR>
<TR>
    <TD COLSPAN="2">
        <INPUT TYPE="button" VALUE="Click to Submit"
            onClick="errorCheck()"
    </TD>
</TR>
</TABLE>
</FORM>

```

Observe que al final del formulario los programadores ya han insertado un manipulador de eventos onClick en el botón Submit, desde el que llaman a la función errorCheck(). Ya que se tomaron las molestias de introducirla en el manipulador, la usaremos para el nombre de la función que creamos.

Comprobación de formularios en busca de campos de texto vacíos

Lo siguiente es la primera parte del código JavaScript que utilizaremos para comprobar la información del visitante:

```

<SCRIPT LANGUAGE = "JavaScript">
<!-- El código tras esto será ignorado por los navegadores
    más antiguos
//Creación de la función errorCheck
function errorCheck()
{
    // Creación de algunas variables para el script
    var requiredFieldsErrorMessage = "";
    var firstNameEntered = "";
    var lastNameEntered = "";
    var emailEntered = "";
    var areaCodeEntered = "";
}
</pre>

```

Validación de errores en los formularios

```
var zipCodeEntered = "";
var pullDownErrorMessage = "";
var emailErrorMessage = "";
var areaCodeErrorMessage = "";
var zipCodeErrorMessage = "";
...
}

// Dejamos de ocultar código aquí -->
</SCRIPT>
```

En el código anterior establecimos algunas variables. Utilizaremos las cuatro primeras en este proyecto y las demás en proyectos posteriores. Estas variables seguirán la pista de todos los mensajes de error específicos que genere el usuario. Al final del *script*, todos los mensajes de error aparecerán juntos y un mensaje de error más comprensivo saludará al visitante del sitio.

Después, comenzaremos a añadir la parte de la función que comprobará la información.

```
<SCRIPT LANGUAGE = "JavaScript">
<!--El código tras esto será ignorado por los navegadores
  más antiguos
// Creación de la función errorCheck
{
  // Creación de algunas variables para el script
  var requiredFieldsErrorMessage = "";
  var firstNameEntered = "";
  var lastNameEntered = "";
  var emailEntered = "";
  var areaCodeEntered = "";
  var zipCodeEntered = "";
  var pullDownErrorMessage = "";
  var emailErrorMessage = "";
  var areaCodeErrorMessage = "";
  var zipCodeErrorMessage = "";
}

// Dejamos de ocultar código aquí --&gt;
&lt;/SCRIPT&gt;</pre>
```

```
firstNameEntered = document.forms['promotion'].elements['firstName'].value
lastNameEntered = document.forms['promotion'].elements['lastName'].value
emailEntered = document.forms['promotion'].elements['email'].value
...
}

// Dejamos de ocultar código aquí -->
</SCRIPT>
```

Pediremos al visitante que introduzca la información en tres campos: nombre, apellidos y dirección de correo electrónico. En capítulos anteriores aprendimos a acceder a los valores desde los elementos FORM en una página HTML. Aquí utilizaremos métodos parecidos para obtener los valores de los tres campos que necesitamos comprobar. Además, la línea:

```
firstNameEntered =
document.forms['promotion'].elements['firstName'].value
```

mira lo que el visitante introdujo en el campo de texto destinado al nombre y coloca ese valor en la variable *firstNameEntered*. Esta línea va seguida de sentencias similares para los otros dos campos. Si el usuario no introduce información en estas tres solicitudes, las variables estarán vacías. Después, estableceremos una sentencia if para comprobar estos valores.

```
<SCRIPT LANGUAGE = "JavaScript">
<!--El código tras esto será ignorado por los navegadores
  más antiguos
// Creación de la función errorCheck
function errorCheck()
{
}</pre>
```

```
// Creación de algunas variables para nuestro script
var requiredFieldsErrorMessage = "";
var firstNameEntered = "";
var lastNameEntered = "";
var emailEntered = "";
var areaCodeEntered = "";
var zipCodeEntered = "";
var pullDownErrorMessage = "";
var emailErrorMessage = "";
var areaCodeErrorMessage = "";
var zipCodeErrorMessage = "";

// Campos necesarios para la comprobación de errores
firstNameEntered = document.forms['promotion'].elements['firstName'].value
lastNameEntered = document.forms['promotion'].elements['lastName'].value
emailEntered = document.forms['promotion'].elements['email'].value
if ((!firstNameEntered) || (!lastNameEntered) ||
(!emailEntered))
{
    if (!firstNameEntered)
    {
        requiredFieldsErrorMessage = "- your first
name\r"
    }
    if (!lastNameEntered)
    {
        requiredFieldsErrorMessage =
requiredFieldsErrorMessage +
"- your last
name\r"
    }
    if (!emailEntered)
    {
        requiredFieldsErrorMessage =
requiredFieldsErrorMessage +
address\r"
    }
}
```

}

// Dejamos de ocultar código aquí -->

</SCRIPT>

La primera línea de la sentencia if es la siguiente:

if (!!firstNameEntered) || (!!lastNameEntered) ||
(!emailEntered))

Dicha línea indica al motor de ejecución de JavaScript lo siguiente: si un visitante no introduce un nombre, unos apellidos o una dirección de correo electrónico, la sentencia if es falsa. La doble barra vertical (||) es el signo utilizado para la disyuntiva OR en sentencias condicionales. Al utilizar un operador OR en una sentencia condicional, como puede ser una sentencia if, si se cumple cualquiera de las condiciones, la sentencia if es verdadera (true) y se ejecutan las sentencias siguientes. También hemos utilizado el operador ! en la sentencia if anterior. En JavaScript (y la mayoría de los lenguajes de programación) éste es el símbolo de la negación ("no"). Por tanto, la línea anterior de código dice al motor que devuelva un valor de verdadero (true) si alguna de las variables no tiene un valor. El motor ejecutará las siguientes sentencias. Si los tres campos están rellenos correctamente, el script saltará por delante del código que sigue a la sentencia if.

Hay posibilidades de que alguno de los tres campos no se haya llenado correctamente, de modo que insertamos algo de código que se ejecutará si se produce algún error. Añadimos tres sentencias if más que comprueban los tres campos individualmente y, si fuera necesario, almacenarán un breve mensaje en la variable requiredFieldsErrorMessage que utilizaremos después para decir al usuario que hay un campo erróneo.

if (!!firstNameEntered)

La sentencia if anterior implica que si el visitante no introduce un nombre, se ejecuta la siguiente línea:

```
requiredFieldsErrorMessage = "- your first name\r";
```

Esta línea asigna la cadena “- your first name” a la variable requiredFieldsErrorMessage. Observará que aparece \r al final de la cadena (como se indicó en capítulos anteriores, la barra invertida sirve para crear caracteres de escape, indicando al motor JavaScript que ejecute lo que representa el carácter que acompaña a la barra). El valor r le indica al motor que inserte un retorno de línea, similar a presionar la tecla Intro. Hay dos sentencias if más que comprueban la presencia de un valor en los otros dos campos en los que estamos interesados, y que añadirá información a la variable requiredFieldsErrorMessage si están vacíos.

```
"- your last name\r"; requiredFieldsErrorMessage +
```

La sentencia anterior se ejecuta si la comprobación del campo de los apellidos determina que está vacío. Añadimos al mensaje de error ya existente del visitante por qué puede cometer más de un error. Por tanto, a la derecha de la expresión, el script le indica al motor que añada last name\r al valor que ya existe en requiredFieldsErrorMessage, colocando ambos en la variable requiredFieldsErrorMessage, si se comete más de un error, queremos enumerar ambos, no sólo sustituir el primer error por el siguiente. El tercer bucle while sigue la misma sintaxis y comprueba la presencia de un valor en el campo de la dirección de correo electrónico.

Cómo asegurarse de que se selecciona una opción del menú desplegable

Después, tenemos que asegurarnos de que el usuario seleccione una opción del menú desplegable que indique lo

que quiere ver en una versión *on-line* de la revista. Afortunadamente, el código que utilizaremos para conseguir esto es muy parecido al código que ya hemos usado para probar los valores de los campos de texto que contienen el nombre del usuario. Insertaremos el código JavaScript para comprobar el menú desplegable siguiendo el código que creamos para comprobar los cuadros de texto.

```
// Comprobación de errores en los desplegables
var selectPos = document.forms['promotion']
    .elements['whatTheyWant'].selectedIndex;
if (document.forms['promotion'].elements['whatTheyWant']
    .options[selectPos].value == 'default')
{
    pullDownErrorMessage = "- you didn't tell us what you
    want to see in Stitch";
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

Ya que sólo vamos a comprobar un campo del formulario, podemos usar una sola sentencia if. En la sentencia if anterior comparamos el valor del campo whatTheyWant con el valor predeterminado. Si observa el código HTML del menú desplegable, verá que default es el valor asignado a la primera opción del menú. Por tanto, si el usuario no modifica su posición inicial y selecciona otra opción, el valor seguirá siendo default y el script asignará un breve mensaje a la variable pullDownErrorMessage.

Además, hemos escrito un script que comprueba si algunos de los campos del formulario están vacíos; sin embargo, hay casos en los que esto no es suficiente información. En ciertas instancias sería útil comprobar una respuesta falsa o errónea por parte del usuario. El siguiente proyecto cubrirá varios ejemplos de tales casos.

Proyecto II: cómo comprobar datos erróneos en los campos de un formulario

Comprobación de caracteres correctos en las direcciones de correo electrónico

Nunca puede estar seguro al 100% de que alguien le dará una dirección electrónica correcta. No hay fórmula o base de datos central que tenga registradas todas las direcciones electrónicas correctas del mundo. Sin embargo, puede asegurarse de que alguien le dará algo que al menos parezca una dirección electrónica comprobando la existencia del carácter @ y un punto. Incluso aunque vamos a tratar esta sección como un nuevo proyecto, el código de este capítulo es realmente parte de un *script*. Añadiremos este nuevo código después del código ya existente, que está todavía dentro de la función *errorCheck()*.

```
// Comprobación de errores en la dirección de correo
// electrónico
if (emailEntered)
{
    emailEntered = document.forms['promotion']
    .elements['email'].value;
    emailValue = new String (emailEntered);
    emailHasAt = emailValue.indexOf("@");
    emailHasPeriod = emailValue.indexOf(".");
    if ((emailHasAt == -1) || (emailHasPeriod == -1))
    {
        emailErrorMessage = "-your email address\r";
    }
}
// Dejamos de ocultar código aquí ...
</SCRIPT>
```

Lo primero que hemos hecho aquí es retomar de nuevo el valor del campo en cuestión y asignarlo a una variable. Entonces creamos un nuevo objeto llamado *emailValue* (hay otros *IMAGE* que utilizamos en el Capítulo 2 para los *rollovers* de las imágenes). Sin embargo, en lugar de crear un objeto *IMAGE*, creamos un objeto *STRING* que almacenará el valor que está ahora en la variable *emailEntered*:

```
emailValue = new String (emailEntered);
```

La razón por la que vamos a introducir el valor del campo en un objeto en lugar de sencillamente guardarlo en una variable normal es porque los objetos son más versátiles y nos permiten acceder a métodos que no podemos utilizar en variables. Utilizaremos el método *indexOf()* en este *script*. Esto nos permite buscar caracteres específicos que aparecen en la cadena; en este caso, buscaremos el carácter @ y un punto.

```
emailHasAt = emailValue.indexOf("@");
emailHasPeriod = emailValue.indexOf(".");
```

La primera línea busca el carácter @ en el objeto *emailValue* y recuerda cuál es su posición. Por ejemplo, si *emailValue* fuera "dan@wire-man.com", el carácter @ estaría en la posición 3 (recuerde, JavaScript comienza contando desde 0); por tanto, *emailHasAt* contendría el valor 3. Si el carácter no se encontrara, obtendría el valor -1.

La segunda línea es básicamente lo mismo, pero comprueba la existencia de un punto y almacena lo que encuentra en la variable *emailHasPeriod*. Así, las siguientes líneas:

```
if ((emailHasAt == -1) || (emailHasPeriod == -1))
{
    emailErrorMessage = "-your email address\r";
```

indican al motor que cree un mensaje de error llamado `emailErrorMessage` si la dirección electrónica no tiene un carácter @ o un punto. Despues, comprobaremos los campos de los números telefónicos para asegurarnos de la presencia de un prefijo.

Comprobación de la existencia del prefijo en los números telefónicos

Aquí engañaremos un poco: volveremos a formatear el código HTML del formulario de modo que sólo pueda obtener la información que quiere del campo del número de teléfono (véase la Figura 4.2). Recuerde este truco cuando vaya a crear *scripts* más adelante; a menudo existen formas de manipular el código HTML para desarrollar *scripts* más eficientes. Aquí está el nuevo código HTML para los campos de los números de teléfono:

```
<INPUT TYPE="text" NAME="areaCode" SIZE="3"
MAXLENGTH="3">
<INPUT TYPE="text" NAME="phoneNumber1" SIZE="3"
MAXLENGTH="3">.
< INPUT TYPE="text" NAME="phoneNumber2" SIZE="4"
MAXLENGTH="4">
```

Dividiendo el número de teléfono en tres campos diferentes podemos aislar el prefijo más fácilmente. De esta forma, no tiene que mirar el número de teléfono entero y averiguar qué parte es el prefijo.

Esto puede ser un problema porque los usuarios escriben los códigos de área de forma diferente; por ejemplo, (925)555-1212, 925-555-1212, 925/555.1212, etc. Recuerde esto cuando escriba *scripts* (hay otras formas aparte de las soluciones JavaScript que son más fáciles de implementar). No tema explorar estas otras posibilidades.

Figura 4.2
Página del formulario de promoción de Stitch con nuevos campos para los números de teléfono.

El número de teléfono no es un campo obligatorio; sin embargo, si un visitante ofrece voluntariamente un número

de teléfono, querrá asegurarse de que él o ella escriben el prefijo; después de todo, esto es la Web y usted no tiene idea desde dónde está llamando el visitante. (Ignoraremos los números de teléfono internacionales a propósito por ahora. Aunque el ejemplo sigue el modelo de EE.UU., es fácil modificarlo para adaptarlo al formato telefónico de cualquier otro país.)

Ya que hemos decidido utilizar tres campos diferentes para el número de teléfono, podemos comprobar los valores siguiendo el mismo método que para la comprobación de los campos de nombre, apellidos y dirección de correo electrónico.

```
// Comprobación del prefijo
areaCodeEntered =
document.forms['promotion'].elements['areaCode'].value
phoneNumber1Entered =
document.forms['promotion'].elements['areaCode'].value
...
}

// Dejamos de ocultar código aquí -->
</SCRIPT>
```

En las dos primeras líneas del código anterior asignamos los valores del campo del prefijo y del campo del primer número de teléfono a variables. En las siguientes líneas utilizaremos una sentencia if que nos indicará si existe lo que es obligatorio.

```
// Comprobación del prefijo
areaCodeEntered =
document.forms['promotion'].elements['areaCode'].value
phoneNumber1Entered =
document.forms['promotion'].elements['areaCode'].value
.value
if ((!areaCodeEntered) && (phoneNumber1Entered))
```

```
{
    areaCodeErrorMessage = "-please enter your area
    code!r"
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

No habíamos visto antes el operador `&&`. Se trata de otro operador lógico, al igual que el operador `||` que utilizamos anteriormente en el script. El operador `&&` significa que se tienen que cumplir ambas condiciones para que se devuelva el valor true en la sentencia. Por tanto,

```
if (!areaCodeEntered) && (phoneNumberEntered))
```

indica al motor que ejecuta JavaScript que si la variable `areaCodeEntered` no tiene un valor, pero `phoneNumberEntered` sí lo tiene, se ejecutarán las siguientes líneas. De esta manera, sólo se genera un mensaje de error si el visitante introduce un número de teléfono sin un prefijo. Ahora, vayamos al último campo que tenemos que comprobar.

Cómo asegurarse de que los códigos postales sólo tienen números

El código JavaScript para asegurarse de que el código postal del visitante sólo contiene números es más complicado que lo que ya hemos visto, y presenta nuevas características y funciones. El plan global consiste en estudiar la cadena de cinco caracteres almacenada en la variable `zipCodeEntered`, un carácter cada vez, y ver si cada carácter es realmente un número. Si lo es, entonces configuraremos la variable a yes y pasaremos al siguiente carácter. Sin embargo, si encontramos un carácter que no es un número, crearemos un mensaje de error. Esta parte del script co-

mienza como la mayoría de los demás, tomando el valor del campo que necesitamos comprobar y asignándolo a una variable.

```
// Comprobación del código postal
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
...
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

Después, ya que no se pide al usuario que introduzca una dirección, utilizaremos una sentencia if para ver si hay algo en el campo del código postal.

```
// Comprobación del código postal
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
...
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

Si no hay un valor en el campo zipCode, el script ignorará el resto del código postal y pasará a la parte siguiente. Sin embargo, si hay un valor en el campo, necesitaremos crear algunas variables y asignar a algunas de ellas un valor base, postal apropiada.

```
// Comprobación del código postal
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
```

```
if (zipCodeEntered)
{
numberCounter = 0;
zipCounter = 0;
foundNumber = "";
...
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

Hemos creado dos contadores que vamos a utilizar en bucles while, y una tercera variable que utilizaremos para almacenar la salida de nuestra comprobación. Nuestro siguiente paso es crear dos bucles while.

```
// Comprobación del código postal
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
numberCounter = 0;
zipCounter = 0;
foundNumber = "";
while (zipCounter < 5)
{
...
zipCounter++;
}
...
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

El script anterior proporciona las bases de nuestro primer bucle while. El script indica al motor que ejecute las sentencias que siguen mientras la variable zipCounter sea menor de 5. En este punto, sólo hay una sentencia dentro del bucle, pero añadiremos más en un momento. La sentencia que hay ahora, sin embargo, es la clave para asegurarse de que no vamos a crear un bucle infinito.

```
zipCounter++;
```

La línea anterior toma el valor de zipCounter y le añade 1, de forma que cada vez que se ejecute el bucle, zipCounter tendrá un valor con una unidad más que en el paso anterior, hasta que alcance 5 y la condición ya no se cumpla y el script pueda seguir.

Este primer bucle while lo utilizaremos para comprobar los cinco caracteres del campo del código postal. Añadiremos un segundo bucle while dentro del primero que utilizará para efectuar un ciclo por los 10 dígitos (0-9).

```
// Comprobación del código postal
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
    numberCounter = 0;
    zipCounter = 0;
    foundNumber = "";
    while (zipCounter < 5)
    {
        while (numberCounter<10)
        {
            ...
            numberCounter++;
        }
    }
}
```

```
if (foundNumber != "yes" )
{
    zipCounter = 6;
}
else
{
    zipCounter++;
    numberCounter = 0;
}
}
...
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

Ahora, el segundo bucle while preguntará por los números 0-9 cada vez que se ejecute el primer bucle. Por ahora, ignoraremos la sentencia if situada al final del primer bucle y añadiremos las sentencias que necesitamos en el segundo bucle.

```
// Comprobación del código postal
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
    numberCounter = 0;
    zipCounter = 0;
    foundNumber = "";
    while (zipCounter < 5)
    {
        while (numberCounter<10)
```

```

{
    if (zipCodeEntered.substr(zipCounter, 1) ==
        numberCounter)
    {
        foundNumber = "yes";
        numberCounter = 11;
    }
    else
    {
        foundNumber = "no";
    }
    numberCounter++;
}

if (foundNumber != "yes")
{
    zipCounter = 6;
}
else
{
    zipCounter++;
    numberCounter = 0;
}
}

...
}

// Dejamos de ocultar código aquí -->
</SCRIPT>

```

En el código anterior, añadimos una sentencia `if` en el interior del segundo bucle `while`. Esta sentencia `if` es la sección

mas importante de esta parte del `script` (realiza la comparación de los valores del campo del código postal, de modo que veámosla con más detalle). Primero, configuramos la comprobación del valor.

```
if (zipCodeEntered.substr(zipCounter, 1) == numberCounter)
```

Utilizaremos el método `substr()` en esta sentencia para preguntar por un carácter específico del campo del código postal. Cuando utilizamos este método en el Capítulo 1, especificamos el rango determinado de caracteres que queríamos evaluar; en este ejemplo, sólo vamos a comprobar un carácter y utilizaremos el valor de la variable `zipCounter` para indicar al método `substr()` qué carácter queremos evaluar.

Mientras el primer bucle se ejecuta, añadiendo 1 a esta variable en cada ejecución de dicho bucle, comprobamos los caracteres que están en las posiciones 0-4 de la variable `zipCodeEntered`. En otras palabras, comprobaremos los cinco caracteres del código postal.

Vamos a comparar estos valores de uno en uno con el valor de la variable `numberCounter`. Ésta es la variable que es la base de la evaluación del segundo bucle, y mientras el segundo bucle siga existiendo, `numberCounter` verá incrementado su valor en 1 en cada pasada. Así, mientras se ejecuta el segundo bucle, se efectúa una evaluación para ver si el carácter localizado en la posición de `zipCounter` es el mismo que el valor de `numberCounter`, el cual está efectuando un ciclo por los dígitos 0-9.

Si se produce una coincidencia, la sentencia `if` pasará a establecer la variable `foundNumber` a `yes` y configurará el valor de `numberCounter` a 11 de modo que, en su siguiente entrada del segundo bucle `while`, la condición será falsa y el `script` continuará. Si no encuentra una coincidencia, configurará el valor de `foundNumber` a `no`, añadirá 1 al valor de `numberCounter` y ejecutará el segundo bucle de nuevo hasta que encuentre una coincidencia.

Una vez que el segundo bucle ejecuta sus diez iteraciones o se encuentra una coincidencia con un número, el script continuará con la sentencia if que añadimos al primer bucle while anteriormente. Esta sentencia if evalúa el valor de foundNumber. Si en el segundo bucle while no se encuentra un número, configurará zipCounter a 6, lo que nos sacará del primer bucle while en el siguiente paso y creará un mensaje de error. Si se encuentra un número en el segundo bucle, la sentencia if añadirá 1 al valor de zipCounter y permitirá seguir comprobando el siguiente carácter del campo del código postal.

La adición final a esta sección de código será crear otro mensaje de error si se encuentra un carácter no numérico en el campo del código postal.

```
// Comprobación del código postal
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
    numberCounter = 0;
    zipCounter = 0;
    foundNumber = "";
    while (zipCounter < 5)
    {
        while (numberCounter<10)
        {
            if (zipCodeEntered.substr(zipCounter, 1) ==
                numberCounter)
            {
                foundNumber = "yes";
                numberCounter = 11;
            }
            else

```

```

        {
            foundNumber = "no";
        }
        numberCounter++;
    }
    if (foundNumber != "yes")
    {
        zipCounter = 6;
    }
    else
    {
        zipCounter++;
        numberCounter = 0;
    }
}
if (foundNumber != "yes")
{
    zipCodeErrorMessage = "-zip code has non-numbers
in it\r";
}
...
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

En esta sección hemos creado un código bastante difícil y avanzado. Sin embargo, ¡buenas noticias! ¡Casi hemos terminado! Lo único que nos queda por hacer es comprobar y ver si los mensajes de error están creados y mostrarlos al usuario o, si no están creados, enviar el formulario al script CGI.

Proyecto III: cómo permitir que el usuario sepa qué es erróneo

Esta parte final del código no es nueva; vamos a utilizar métodos que se cubrieron anteriormente en el capítulo. Si el mundo fuera un lugar perfecto, el usuario habría llenado el formulario apropiadamente. Sin embargo, éste no siempre es el caso, de modo que ahora que hemos repasado y creando las comprobaciones necesarias para los datos que faltan o que son erróneos, necesitamos transmitir al usuario lo que hemos encontrado. Utilizaremos una sola sentencia `if` para realizar esto.

```
// Poner en conocimiento del usuario lo que falta
if ((requiredFieldsErrorMessage)
   || (whatTheyWantErrorMessage)
   || (areaCodeErrorMessage) || (emailErrorMessage)
   || (zipCodeErrorMessage))
{
    ...
}
else
{
    ...
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

La primera línea comprueba si las variables de los mensajes de error tienen un valor. Si lo tienen, significa que el usuario ha omitido algo y debemos escribir un mensaje de error global. Para ello, primero asignamos el principio de un mensaje a la variable `alertMessage` y después añadimos al final cualquier mensaje de error generado por las distintas

```
// Poner en conocimiento del usuario lo que falta
if ((requiredFieldsErrorMessage)
   || (whatTheyWantErrorMessage)
   || (areaCodeErrorMessage) || (emailErrorMessage)
   || (zipCodeErrorMessage))
{
    alertMessage = "Oops! There's a little trouble with
    the information you've provided\r";
    // construir el resto del mensaje de alerta
    alertMessage = alertMessage +
    requiredfieldsErrorMessage;
    alertMessage = alertMessage +
    whatTheyWantErrorMessage;
    alertMessage = alertMessage + emailErrorMessage;
    alertMessage = alertMessage + areaCodeErrorMessage;
    alertMessage = alertMessage + zipCodeErrorMessage;
    ...
}
else
{
    ...
}
// Dejamos de ocultar código aquí -->
</SCRIPT>
```

Una vez que añadimos otros mensajes de error al que tenemos al principio, todo lo que necesitamos hacer es utilizar la función `alert()` para mostrar el mensaje completo al usuario.

```
// Poner en conocimiento del usuario lo que falta
if ((requiredFieldsErrorMessage)
   || (whatTheyWantErrorMessage) || (emailErrorMessage)
   || (areaCodeErrorMessage) || (zipCodeErrorMessage))
{
}
```

```

alertMessage = "Oops! There's a little trouble with
the information you've provided\r";
// construir el resto del mensaje de alerta
alertMessage = alertMessage +
requiredFieldsErrorMessage;
alertMessage = alertMessage +
whatTheyWantErrorMessage;
alertMessage = alertMessage + emailErrorMessage;
alertMessage = alertMessage + areaCodeErrorMessage;
alertMessage = alertMessage + zipCodeErrorMessage;
alert (alertMessage);
}
else
{
...
}
// Dejamos de ocultar código aquí -->
</SCRIPT>

```

Si no hay nada erróneo en sus respuestas, la sentencia `el-`
`script` CGI.

```

// Poner en conocimiento del usuario lo que falta
if ((requiredFieldsErrorMessage)
|| (whatTheyWantErrorMessage) || (emailErrorMessage)
|| (areaCodeErrorMessage) || (zipCodeErrorMessage))
{

alertMessage = "Oops! There's a little trouble with
the information you've provided\r";
// construir el resto del mensaje de alerta
alertMessage = alertMessage +
requiredFieldsErrorMessage;
alertMessage = alertMessage +
whatTheyWantErrorMessage;

```

```

alertMessage = alertMessage + emailErrorMessage;
alertMessage = alertMessage + areaCodeErrorMessage;
alertMessage = alertMessage + zipCodeErrorMessage;
// Ahora, visualizar el mensaje de error
alert (alertMessage);

}
else
{
document.forms['promotion'].submit();
}
// Dejamos de ocultar código aquí -->
</SCRIPT>

```

ANÁLISIS DEL SCRIPT

Si esto no fuera, digamos, una página repleta de código JavaScript, no sabríamos lo que es. Sin embargo, ahora tenemos un *script* que hará que el personal de marketing cancele sus alabanzas a su jefe. Miremos el *script* una última vez y revisemos lo que realizamos.

1. Primero, creamos algunas variables que almacenan los valores que el usuario ha introducido en los campos necesarios, así como variables para almacenar los mensajes de error que se deban crear.

```

<SCRIPT Language="JavaScript">
<!-- El código tras esto será ignorado por los navegadores
    más antiguos
    // Creación de la función errorCheck
    function errorCheck()
    {
        // Creación de algunas variables para el script
        var requiredFieldsErrorMessage = "";
        var firstNameEntered = "";
        var lastNameEntered = "";
    }
</pre>

```

```
var emailEntered = "";
var areaCodeEntered = "";
var zipCodeEntered = "";
var pullDownErrorMessage = "";
var emailErrorMessage = "";
var areaCodeErrorMessage = "";
var zipCodeErrorMessage = "";
```

2. Despues, comenzamos a crear la parte de la función que comprueba las omisiones o errores en las respuestas del usuario e imprime un mensaje de error que permite a éste conocer su equivocación.

El primer paso es almacenar las respuestas del usuario a los campos obligatorios en nuevas variables.

```
// Comprobación de errores en los campos obligatorios
firstNameEntered =
document.forms['promotion'].elements['firstName'].value;
lastNameEntered =
document.forms['promotion'].elements['lastName'].value;
emailEntered =
document.forms['promotion'].elements['email'].value;
```

3. Despues, configuramos una serie de sentencias if para asegurarnos de que las variables contienen datos, y si no, redFieldsErrorMessage.

```
if ((!firstNameEntered) || (!lastNameEntered) ||
(!emailEntered))
{
    if (!firstNameEntered)
    {
        requiredFieldsErrorMessage = "- your first
name\r";
    }
    if (!lastNameEntered)
    {
        requiredFieldsErrorMessage = "- your last
name\r";
    }
    if (!emailEntered)
    {
        requiredFieldsErrorMessage = "- your email
address\r";
    }
}
```

requiredFieldsErrorMessage = "- your first name\r";

requiredFieldsErrorMessage = "- your last name\r";

requiredFieldsErrorMessage = "- your email address\r";

```
requiredFieldsErrorMessage =
requiredFieldsErrorMessage + "- your last name\r";
}
if (!emailEntered)
{
    requiredFieldsErrorMessage =
requiredFieldsErrorMessage + "- your email
address\r";
}
}
```

4. Entonces establecemos una sentencia if para asegurarnos de que el usuario selecciona una opción distinta a la opción predeterminada del menú desplegable. Si el usuario no selecciona otra opción, el script asigna un mensaje corto a la variable pullDownErrorMessage.

```
// Comprobación de errores en los desplegables
if (document.forms['promotion'].elements['whatTheyWant']
.value == 'default')
{
    pullDownErrorMessage = "- you didn't tell us what you
want to see in Stitch";
}
}
```

5. El siguiente paso de la función errorCheck() es establecer una serie de sentencias if que evalúen la cadena introducida en el campo de dirección de correo electrónico del formulario y comprueben la presencia de un carácter @ y un punto.

```
// Comprobación de errores en la dirección de correo
// electrónico
if (emailEntered)
{
    emailEntered = document.forms['promotion']
.elements['email'].value;
}
```

emailEntered = document.forms['promotion'].elements['email'].value;

```

emailValue = new String (emailEntered);
emailHasAt = emailValue.indexOf("@");
emailHasPeriod = emailValue.indexOf(".");
if ((emailHasAt == -1) || (emailHasPeriod == -1))
{
    emailErrorMessage = "-your email address|r";
}

```

6. Si el usuario introduce un número de teléfono, debemos asegurarnos de que también ha introducido un prefijo para ese número. Por tanto, configuramos una serie de sentencias if que evalúan los valores almacenados en ambos campos del formulario.

```

// Comprobación del prefijo
areaCodeEntered =
document.forms['promotion'].elements['areaCode'].value;
phoneNumber1Entered =
document.forms['promotion'].elements['phoneNumber1'].value;
if ((!areaCodeEntered) && (phoneNumber1Entered))
{
    areaCodeErrorMessage = "-please enter your area
code|r";
}

```

7. Después, nos aseguramos de que el campo del código postal sólo contiene números. Establecemos sentencias if y dos bucles while para examinar los caracteres individuales de la cadena introducida en el campo del código postal y comprobar que son números enteros.

```

// Comprobación del código postal
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
// no queremos hacer nada si el visitante
// no ha introducido nada en el código postal

```

```

if (zipCodeEntered)
{
    numberCounter = 0;
    zipLength = 5;
    zipCounter = 0;
    foundNumber = "";
    while (zipCounter < 5)
    {
        while (numberCounter<10)
        {
            if (zipCodeEntered.substr(zipCounter, 1) ==
                numberCounter)
            {
                foundNumber = "yes";
                numberCounter = 11;
            }
            else
            {
                foundNumber = "no";
            }
            numberCounter++;
        }
        if (foundNumber != "yes")
        {
            zipCounter = 6;
        }
        else
        {
            zipCounter++;
            numberCounter = 0;
        }
    }
    if (foundNumber != "yes")

```

```

{
    zipCodeErrorMessage = "-zip code has non-numbers
    in it\r";
}
}

8. Finalmente, configuramos algunas sentencias if para evaluar las diversas variables de mensajes de error. Si almacenan un valor, añadimos ese valor a la variable alertMessage. Una vez que todos los mensajes de error están comprobados, imprimimos el mensaje final para que el usuario lo vea. Si no se encuentran mensajes de error, llamamos a la función submit() del formulario.

// Poner en conocimiento del usuario lo que falta
if ((requiredFieldsErrorMessage)
    || (pullDownErrorMessage) || (emailErrorMessage)
    || (areaCodeErrorMessage) || (zipCodeErrorMessage))
{
    alertMessage = "Oops! There's a little trouble with
    the information you've provided\r";
    // construir el resto del mensaje de alerta
    alertMessage = alertMessage +
    requiredFieldsErrorMessage;
    alertMessage = alertMessage + pullDownErrorMessage;
    alertMessage = alertMessage + emailErrorMessage;
    alertMessage = alertMessage + areaCodeErrorMessage;
    alertMessage = alertMessage + zipCodeErrorMessage;
    // Ahora, visualizar el mensaje de error
    message alert ( alertMessage )
}

else // no hay problemas con el formulario
{
    // enviar los datos de este formulario al CGI
    // document.forms['promotion'].submit()
}

```

Hechos aprendido nuevos conceptos en este capítulo y mejorado nuestro conocimiento de conceptos que ya habíamos discutido. Veamos lo que hemos cubierto:

- La jerarquía JavaScript y nuestra capacidad de acceder y manipular sus objetos (especialmente objetos de formulario).
- Sentencias if y bucles while.
- El uso del método alert() para ofrecer información al usuario.
- Operadores lógicos: ||, && y !.
- Introducción de un retorno de carro utilizando \r.
- El objeto STRING y algunas de sus propiedades.

Resumen

Cubrimos mucho terreno en este capítulo. Si puede memorizar estos conceptos y estructuras, está bien encaminado hacia el dominio de JavaScript. Entender y utilizar las sentencias if y los bucles while es la clave para las soluciones avanzadas de JavaScript (a medida que utilice JavaScript esto será más palpable).

Proyectos avanzados

Aquí hay unas cuantas ideas que puede implementar si desea ir más allá en su entendimiento de los conceptos cubiertos en este capítulo:

1. Intente validar distintos campos del formulario que no se hayan cubierto en este capítulo, como los botones de radio.
2. Evalúe cuadros de texto para respuestas específicas, posiblemente para desarrollar un cuestionario *on-line*.

Ventanas y marcos

JavaScript



En este capítulo

- Una mirada al objeto WINDOW.
- Proyecto I: cómo crear, llenar y cerrar ventanas.
- Proyecto II: cómo utilizar JavaScript para comunicarse entre marcos.
- Resumen.
- Proyectos avanzados.

El reciente conocimiento de JavaScript nos ha permitido hacer cosas bastante interesantes. Hemos pasado los últimos dos capítulos trabajando con formularios y ahora es el momento de cambiar a algo nuevo. Excepto para la página principal, el sitio Stitch se creó utilizando marcos. En este capítulo vamos a trabajar en las páginas secundarias del sitio y a ayudarle a acelerar el trabajo en JavaScript con marcos y ventanas.

Sin embargo, antes de introducirnos en proyectos específicos, vamos a echar un vistazo más de cerca a lo que son las ventanas y los marcos y cuál es su posición en la jerarquía de objetos de JavaScript. Tratar con marcos y múltiples ventanas puede llegar a ser bastante complejo y con fre-

cuencia puede parecer muy enrevesado. Al final de este capítulo esperamos que tenga un sólido entendimiento de cómo las ventanas y marcos, al combinarse con JavaScript, le pueden ayudar a crear cosas interesantes.

Una mirada al objeto WINDOW

El objeto **WINDOW** está en la cima de la jerarquía de JavaScript; el objeto **DOCUMENT** y los objetos por debajo de él son todos descendientes del objeto **WINDOW**. Los proyectos que hemos completado hasta ahora estaban contenidos en una sola ventana, de modo que nunca tuvimos que tratar con el objeto **WINDOW** ni con sus propiedades. Sin embargo, no siempre será éste el caso. Hay muchas ocasiones en las que es ventajoso crear una nueva ventana o acceder a objetos contenidos en otra ventana.

Ya que la ventana es un objeto, podemos acceder a las propiedades y objetos que ésta contiene, así como a sus propiedades. Al utilizar JavaScript, podemos modificar el tamaño de una ventana, configurar las barras de herramientas, acceder al array **HISTORY**, y mucho más. Introduzcámonos en el primer proyecto y veamos lo que significa todo esto.

Proyecto I: cómo crear, llenar y cerrar ventanas

Hay una página en el sitio web de Stitch que enumera las compañías que han puesto anuncios en el número actual de la revista. Stitch es una revista de moda y atrae a grandes compañías anunciantes; los anuncios en sí mismos son con frecuencia tan rompedores como la moda. Ya que la revista está bombardeada con peticiones de los lectores que quieren entrar en contacto con las compañías, la colocación de los anuncios en el sitio web resulta una gran idea. Por tan-

to, han decidido ampliar la funcionalidad de la página y hacer de cada nombre de una compañía un enlace que llevará al usuario a la página con la información de contacto de la compañía, un enlace al sitio web y un enlace a una copia del anuncio en curso.

Adivine quién tiene que hacer esto realidad. Eso es, usted. Sin embargo, no hay razón para ponerse nervioso. Con las recientes adiciones al sitio va a ser conocido como el guru del departamento de JavaScript. Después de una rápida reunión de planificación con el jefe, piense que ha inventado una gran manera de realizar la tarea. El único problema que sigue apareciendo es el hecho de que habrá al menos veinte anunciantes en cada número mensual: tantas páginas añadidas al sitio saturarían el servidor y crearían demasiados archivos que actualizar cada mes. De manera que éste es el plan: utilizaremos arrays de JavaScript en la página de la lista que ya existe para almacenar toda la información que necesitaremos para crear dinámicamente una página para cada compañía, que colocaremos después en una ventana nueva. Esto ahorrará espacio en el servidor, permitirá actualizar un archivo al mes y nos dará un fundamento para el proyecto de este capítulo. Hay tres partes para crear este *script*: primero, definiremos y crearemos los arrays para la información de cada compañía. Después, escribiremos la función que abrirá la ventana nueva y la llenaremos con el contenido. Finalmente, insertaremos los manipuladores de eventos adecuados en el código HTML para hacer que todo suceda.

Creación y definición de arrays

Lo primero es crear los arrays que van a contener la información de la compañía. Será necesario almacenar nueve artículos para cada compañía: nombre de la compañía, calle, ciudad, estado, código, número de teléfono, número de fax, sitio web (si lo tiene) y, finalmente, el URL de la página que

tenga el anuncio de la compañía. En el Capítulo 3 se utilizaron dos arrays para cada grupo, uno con el nombre de la página y otro con el URL. En gran parte, aquí vamos a utilizar la misma idea para almacenar la información de la compañía; sin embargo, en lugar de dos arrays para cada grupo usaremos sólo uno para cada compañía. Mientras que anteriormente asignábamos y hacíamos referencia a cada elemento del array por su posición, en este proyecto vamos a dar a cada elemento un nombre que usaremos para referirnos a él.

Comencemos el *script* definiendo los arrays para las compañías. Ya que esta página no se va a añadir hasta el próximo número, no sabemos qué compañías serán las anunciantes. Mientras tanto, sólo crearemos arrays para dos compañías ficticias, que nos permitirán conseguir que los *scripts* funcionen ahora mismo.

```
<SCRIPT Language="JavaScript">  
  <!-- El código a partir de aquí será ignorado por los  
      navegadores más antiguos  
    // Arrays para la información de las compañías  
      // Definición de los arrays  
      Company1_info = new Array(9);  
      Company2_info = new Array(9);  
      ...  
    // Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Hemos definido dos arrays, uno para Company1 y otro para Company2, y hemos dado a cada uno la longitud de nueve elementos. Después, necesitamos asignar la información de contacto de cada compañía a los arrays. Insertemos la información de Company1 en el array.

```
<SCRIPT Language="JavaScript">  
  <!-- El código a partir de aquí será ignorado por los  
      navegadores más antiguos
```

```
// Arrays para la información de las compañías
// Definición de los arrays
Company1_info = new Array(9);
Company2_info = new Array(9);
// Rellenar el array de la primera compañía
Company1_info['name'] = 'Company 1';
Company1_info['street'] = '1235 company way #1';
Company1_info['city'] = 'Smallsville';
Company1_info['state'] = 'California';
Company1_info['zip'] = '91367';
Company1_info['phone'] = '818-555-1212';
Company1_info['fax'] = '818-555-1213';
Company1_info['website'] = 'http://www.company1.com';
Company1_info['ads'] = 'company1_ads.html';
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Cuando vaya a llenar un array con tantos elementos como los que tenemos en el código anterior, es más fácil asignar a cada elemento un nombre en lugar de utilizar la posición numérica. Es fácil olvidar dónde está localizado un elemento específico si sólo utiliza el número de posición como referencia. El array Company1_info ahora contiene toda la información que necesitaremos para crear una página para Company1 más adelante en el *script*. Rellenemos el array para Company2.

```
<SCRIPT Language="JavaScript">
<!-- El código a partir de aquí será ignorado por los
    navegadores más antiguos
// Arrays para la información de las compañías
// Definición de los arrays
Company1_info = new Array(9);
Company2_info = new Array(9); 121
// Rellenar el array de la primera compañía
Company1_info['name'] = 'Company 1';
Company1_info['street'] = '1235 company way #1';
Company1_info['city'] = 'Smallsville';</pre>
```

```

Company1_info['state'] = 'California';
Company1_info['zip'] = '91367';
Company1_info['phone'] = '818-555-1212';
Company1_info['fax'] = '818-555-1213';
Company1_info['website'] = 'http://www.company1.com';
Company1_info['ads'] = 'company1_ads.html';

// Rellenar el array de la segunda compañía

Company2_info['name'] = 'Company 2';
Company2_info['street'] = '4321 company way #2';
Company2_info['city'] = 'Bigsville';
Company2_info['state'] = 'California';
Company2_info['zip'] = '91235';
Company2_info['phone'] = '818-555-2121';
Company2_info['fax'] = '818-555-2122';
Company2_info['website'] = 'http://www.company2.com';
Company2_info['ads'] = 'company2_ads.html';

// Dejamos de ocultar el código aquí -->

</SCRIPT>

```

Ahora tenemos toda la información que necesitamos almacenada de forma segura en los dos arrays (si necesita recordar lo básico sobre los arrays, consulte el proyecto II del Capítulo 3). Es el momento de ir al siguiente paso.

Creación de la función

La función para este *script* va a tener que realizar varias cosas. Primero, debe decidir qué compañía se eligió y reunir la información procedente del array adecuado. Después, deberá abrir una ventana nueva y, finalmente, llenará dicha ventana con la información del array. Establezcamos primero el marco de trabajo para la función.

```

// Función creadora de la página
function PageCreator(selection)
{
    ...
}

```

```
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

En el código anterior creamos la función `PageCreator()` y configuramos la variable `selection` para aceptar el valor que se pasa a la función desde el manipulador de eventos. Este valor será el nombre de la compañía que seleccionó el usuario. El paso siguiente es averiguar la compañía que seleccionó el usuario y reunir su información.

```
// Función creadora de la página
function PageCreator(selection)
{
    var company = eval(selection + "_info['name']");
    var street = eval(selection + "_info['street']");
    var city = eval(selection + "_info['city']");
    var state = eval(selection + "_info['state']");
    var zip = eval(selection + "_info['zip']");
    var phone = eval(selection + "_info['phone']");
    var fax = eval(selection + "_info['fax']");
    var website = eval(selection + "_info['website']");
    var ads = eval(selection + "_info['ads']");
    ...
}

// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Ahora sabemos que el valor de `selection` va a ser el nombre de la compañía, y hemos usado aquellos nombres en los nombres de los arrays que almacenan la información. En el código anterior utilizamos la función `eval()` para combinar el valor variable con una cadena, que cuando se combina, llamará a la información almacenada en la posición apropiada del array de esa compañía. Una vez que la información se ha recuperado del array, se introduce en una variable que utilizaremos más tarde al imprimir el contenido de la ventana emergente. Este proceso se repite para cada una de las nueve partes distintas de información que vamos a necesitar.

Una vez está la información de la compañía seleccionada y almacenada de forma segura en las variables, abriremos una ventana nueva en la que situar esa información. Echemos un vistazo a la sintaxis del método `open()` del objeto `WINDOW`.

```
window.open(URL, ventanaNombre, ventanaPropiedades)
```

Hay tres elementos que necesita incluir en el método `open()`: el URL de la página que desea situar en la ventana nueva; una cadena con el nombre que desea que la ventana utilice cuando se haga referencia a ella como destino en la etiqueta `<A HREF>`, y una lista opcional de propiedades estándar de la ventana que puede personalizar, como anchura, altura o la inclusión de barras de herramientas.

Una ventana no sólo puede llevar el nombre que se usará como destino, sino que también se puede asociar con un nombre que pueda utilizar al hacer referencia al objeto `WINDOW` que usa JavaScript. Para asignarle este segundo nombre utilice la siguiente sintaxis, donde `Nombre` es el nombre que desea asignar a la ventana.

```
Nombre = window.open(URL, ventanaNombre,  
ventanaPropiedades)
```

Insertemos en el *script* el código que abrirá la ventana.

```
// Función creadora de la página  
function PageCreator(selection)  
{  
  
var company = eval(selection + "_info['name']");  
var street = eval(selection + "_info['street']");  
var city = eval(selection + "_info['city']");  
var state = eval(selection + "_info['state']");  
var zip = eval(selection + "_info['zip']");  
var phone = eval(selection + "_info['phone']");  
var fax = eval(selection + "_info['fax']");  
var website = eval(selection + "_info['website']");  
var ads = eval(selection + "_info['ads']");
```

```
// Abrir la ventana nueva
infowin = window.open("blank.html", "Company_Info",
"menubar=yes, width=250, height=200");
...
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Lo primero que hacemos en el código anterior es asignar a la ventana nueva el nombre `infowin`. Una vez hecho esto, seguimos con el propio método `open()` (el URL de la página que queremos situar en la ventana es `blank.html`). Vamos a llenar la ventana dinámicamente con el contenido de los arrays, de manera que esa página HTML esté tan vacía como el nombre sugiere. El siguiente elemento del código es el nombre de destino al que vamos a asignarlo, `Company_Info`. No queremos usar esto en el `script`, pero es un elemento necesario. El siguiente elemento es la lista de propiedades que queremos que tenga la ventana. Incluiremos la barra de herramientas que contiene los menús, como archivo, editar, ver, etc., y daremos a la ventana una anchura de 250 píxeles y una altura de 200 píxeles.

Ahora que hemos abierto la ventana nueva, debemos llenarla con la información almacenada en las variables. Para hacerlo usaremos el método `document.write()`. Ya hemos utilizado este método en varias ocasiones en este libro, pero ahora vamos a añadir un nuevo giro. Si utilizáramos la sintaxis estándar:

```
document.write('texto que se escribirá');
```

simplemente volveríamos a escribir el contenido en la página del listado de compañías y no en la ventana nueva. Para escribirlo en esta última tenemos que indicar al motor de JavaScript específicamente dónde queremos escribir el código HTML. Esto se puede realizar insertando el nombre de la ventana antes del documento, como sigue:

```
infowin.document.write('texto que se escribirá');
```

Ahora que sabemos cómo escribir en la ventana nueva, añadiremos el código que hará justo eso.

```
// Función creadora de la página
function PageCreator(selection)
{
    var company = eval(selection + "_info['name']");
    var street = eval(selection + "_info['street']");
    var city = eval(selection + "_info['city']");
    var state = eval(selection + "_info['state']");
    var zip = eval(selection + "_info['zip']");
    var phone = eval(selection + "_info['phone']");
    var fax = eval(selection + "_info['fax']");
    var website = eval(selection + "_info['website']");
    var ads = eval(selection + "_info['ads']");

    // Abrir la ventana nueva
    infowin = window.open("blank.html","Company_Info",
    "menubar=yes,width=250,height=200");

    // Escribir contenido en la ventana nueva
    infowin.document.write
    ("<HTML><HEAD><TITLE>Company Information</TITLE>
     </HEAD><BODY BGCOLOR='#FFFFFF'><CENTER>")

    infowin.document.write
    ("<TABLE BORDER='0' CELLSPACING='0'
    CELLPADDING='0'><TR><TD><B>" + company + "</B>
     </TD></TR>");

    infowin.document.write
    ("<TR><TD>" + street + "</TD></TR>");

    infowin.document.write
    ("<TR><TD>" + city + ", " + state + " " + zip +
     "</TD></TR>");

    infowin.document.write
    ("<TR><TD>" + phone - " " + phone + "</TD></TR>
     <TR><TD>fax - " + fax + "</TD></TR>");

    infowin.document.write
    ("<TR><TD><A HREF='" + website + "' TARGET='_TOP'>" +
     website + "</A></TD></TR>");
```

```
infowin.document.write
("<TR><TD ALIGN='CENTER'><BR><A HREF='" + ads + "'"
TARGET='Content'>View the Ad</A></TD></TR> </TABLE>");
infowin.document.write
("<TABLE BORDER='0' CELLSPACING='0' CELLPADDING='0'
WIDTH='249'><TR><TD WIDTH='249' VALIGN='BOTTOM'
ALIGN='RIGHT'><BR>
<A HREF='javascript:window.close()'><IMG
SRC='images/closer.gif'
BORDER='0'></A></TD></TR>");

infowin.document.write
("<TR><TD WIDTH='249'
BGCOLOR='#FFFF00'>&nbsp;</TD></TR></TABLE>");

}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Veamos lo que ocurre en el bloque de código anterior. Aunque se puede parecer mucho, no es igual. Para facilitar la legibilidad, vamos a utilizar múltiples `document.write()`; métodos para escribir código nuevo. En la primera línea creamos un nuevo documento HTML e insertamos las etiquetas HEAD y BODY. En los *scripts* anteriores con `document.write()`, utilizábamos una barra invertida (\) para sustituir las comillas que queríamos imprimir; en este ejemplo, sencillamente vamos a utilizar dos tipos de comillas. Utilizaremos comillas dobles para indicar el principio y el final de lo que queremos imprimir, mientras que dentro de esas comillas usaremos comillas simples cuando queramos que dichos caracteres se impriman realmente en el código. Tanto este método como el método de la barra (\) funcionan de manera eficaz; utilizar uno u otro es cuestión de preferencia personal.

En el resto de las líneas `document.write` utilizamos una combinación de una plantilla de tabla y la información que hemos almacenado en las variables para que se imprima como contenido. Las líneas de la segunda a la última sentencia `document.write` contienen una escritura no tan familiar, de modo que veámoslo más de cerca.

En gran parte, el contenido que vamos a imprimir es bastante sencillo. Sin embargo, si se fija detenidamente, observará algo nuevo.

```
<A HREF='javascript:window.close()'>
```

Puede pensar en el enlace JavaScript en este HREF como en un manipulador de eventos onClick(); cuando el usuario haga clic en el enlace, el código JavaScript que sigue al punto y coma se ejecutará. En este caso, el comando permitirá al usuario cerrar la ventana una vez que él o ella tenga la información necesaria de la página. El cierre de las ventanas es algo que no hemos abordado todavía, de modo que veamos la sintaxis del método close().

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

Por defecto, esta línea cerrará la ventana en la que se encuentre la información. Si el objetivo fuera cerrar otra ventana de un navegador, sencillamente especificaría el nombre, como sigue:

```
infowin.close()
```

Las líneas document.write() son la última parte de la función. La única cosa que queda por hacer en este script es insertar los manipuladores de eventos en el código HTML.

Insertar los manipuladores de eventos

Como en el método close() utilizado anteriormente en el contenido de la ventana emergente, vamos a poner los manipuladores de eventos directamente en las etiquetas HREF del código HTML. Cuando el usuario haga clic en el enlace de una compañía, se llamará a la función PageCreator() y se le pasará el nombre de esa compañía (véase la Figura 5.1). El siguiente código HTML contiene los enlaces con los manipuladores insertados en las etiquetas HREF:

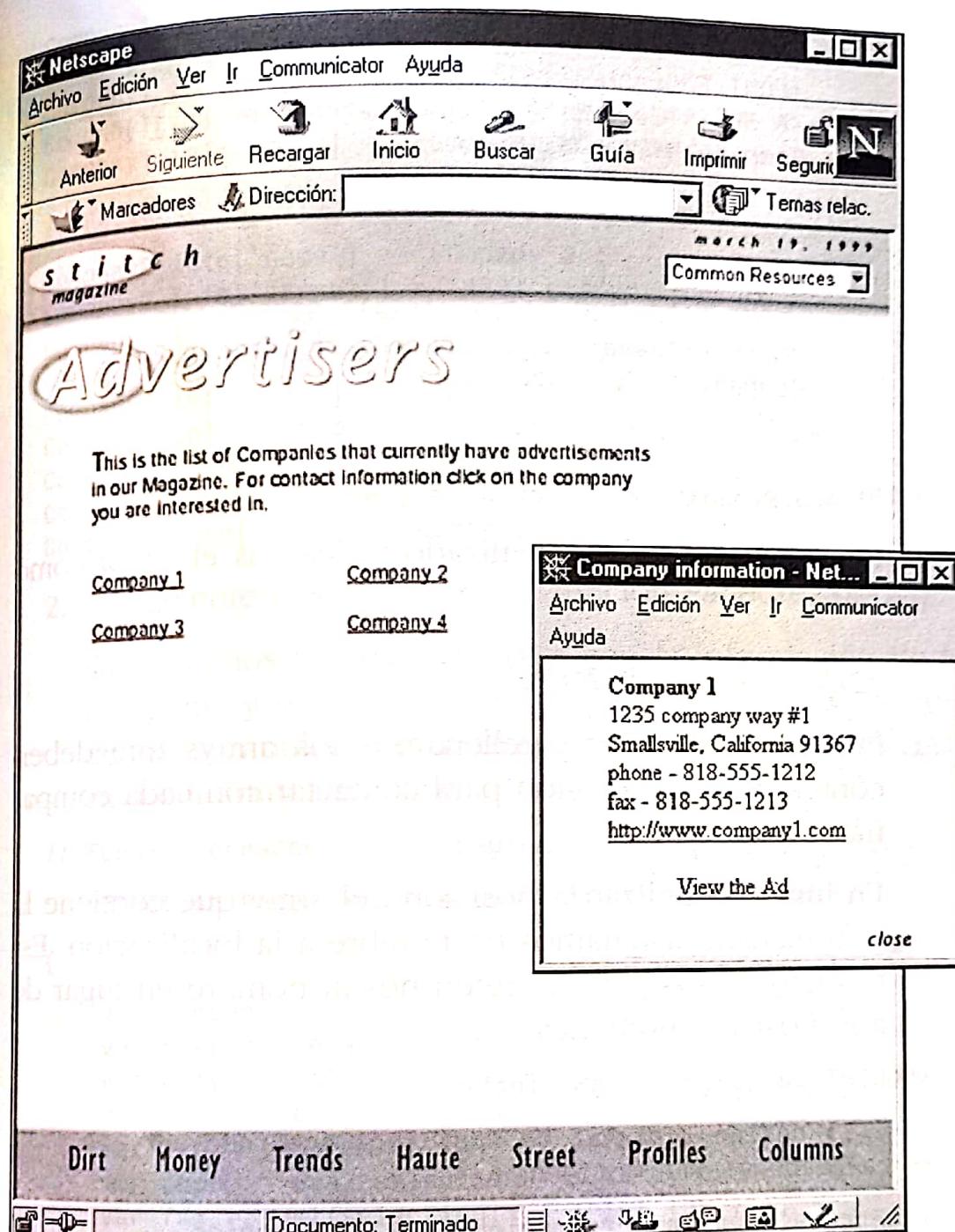


Figura 5.1

La página indexada de anunciantes.

```
<TR>
<TD>
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
</TD>
```

```

<TD>
    <FONT FACE="Helvetica, Arial" SIZE="-1">
        <A HREF="JavaScript:PageCreator('Company1')">
            Company 1</A></FONT>

    </TD>
    <TD>
        <FONT FACE="Helvetica, Arial" SIZE="-1">
            <A HREF="JavaScript:PageCreator('Company2')">
                Company 2</A></FONT>
    </TD>
</TR>

```

Revisemos el trabajo realizado y veamos el *script* como un todo.

ANÁLISIS DEL *SCRIPT*

1. Primero, definimos y rellenamos los arrays que deben contener la información para contactar con cada compañía.

En lugar de utilizar la posición del array que contiene la información, asignamos un nombre a la localización. Este método nos permite referirnos al nombre en lugar de a la posición numérica.

```

<SCRIPT Language="JavaScript">
    <!-- El código a partir de aquí será ignorado por los
        navegadores más antiguos
    // Arrays para la información de las compañías
    // Definición de los arrays
    Company1_info = new Array(9);
    Company2_info = new Array(9);
    // Rellenar el array de la primera compañía
    Company1_info['name'] = 'Company 1';
    Company1_info['street'] = '1235 company way #1';
    Company1_info['city'] = 'Smallsville';
    Company1_info['state'] = 'California';

```

```

company1_info['zip'] = '91367';
Company1_info['phone'] = '818-555-1212';
Company1_info['fax'] = '818-555-1213';
Company1_info['website'] = 'http://www.company1.com';
Company1_info['ads'] = 'company1_ads.html';

// Rellenar el array de la segunda compañía
Company2_info['name'] = 'Company 2';
Company2_info['street'] = '4321 company way #2';
Company2_info['city'] = 'Bigsville';
Company2_info['state'] = 'California';
Company2_info['zip'] = '91235';
Company2_info['phone'] = '818-555-2121';
Company2_info['fax'] = '818-555-2122';
Company2_info['website'] = 'http://www.company2.com';
Company2_info['ads'] = 'company2_ads.html';

```

2. El siguiente paso fue crear la función PageCreator().

Establecimos algunas variables con la información de la compañía y utilizamos el valor pasado a la función desde los manipuladores de eventos para extraer la información del array adecuado.

```

// Función creadora de la página
function PageCreator(selection)
{
    var company = eval(selection + "_info['name']");
    var street = eval(selection + "_info['street']");
    var city = eval(selection + "_info['city']");
    var state = eval(selection + "_info['state']");
    var zip = eval(selection + "_info['zip']");
    var phone = eval(selection + "_info['phone']");
    var fax = eval(selection + "_info['fax']");
    var website = eval(selection + "_info['website']");
    var ads = eval(selection + "_info['ads']");
}

```

3. Una vez introducido el código para extraer la información del array, utilizamos el método open() del objeto WINDOW para crear una ventana nueva con la información de la compañía.

```

infowin=window.open("blank.html","Company_Info",
"menubar=yes,width=250,height=200");

```

4. Con la nueva ventana creada, utilizamos el método write() del objeto DOCUMENT para imprimir el contenido nuevo.

```
infowin.document.write
("<HTML><HEAD><TITLE>Company Information</TITLE>
</HEAD><BODY BGCOLOR='#FFFFFF'><CENTER>")

infowin.document.write
("<TABLE BORDER='0' CELLSPACING='0' CELLPADDING='0'>
<TR><TD><B>" + company + "</B></TD></TR>");

infowin.document.write
("<TR><TD>" + street + "</TD></TR>");

infowin.document.write
("<TR><TD>" + city + ", " + state + " " + zip +
"</TD></TR>");

infowin.document.write
("<TR><TD>" + phone + " " + phone + "</TD></TR>
<TR><TD>fax - " + fax + "</TD></TR>");

infowin.document.write
("<TR><TD><A HREF='" + website + "' TARGET='_TOP'>
" + website + "</A></TD></TR>");

infowin.document.write
("<TR><TD ALIGN='CENTER'><BR><A HREF='" + ads +
"' TARGET='Content'>View the Ad</A></TD></TR>
</TABLE>");

infowin.document.write
("<TABLE BORDER='0' CELLSPACING='0'
CELLPADDING='0' WIDTH='249'><TR><TD WIDTH='249'
VALIGN='BOTTOM' ALIGN='RIGHT'><BR>
<A HREF='javascript:window.close()'>
<IMG SRC='images/closer.gif' BORDER='0'></A>
</TD></TR>");

infowin.document.write
("<TR><TD WIDTH='249' BGCOLOR='#FFFF00'>&nbsp;
</TD></TR></TABLE>");

}

// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

5. El paso final fue insertar los manipuladores de eventos en las etiquetas <A HREF> de cada compañía.

```
<TR>
  <TD>
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
  </TD>
  <TD>
    <FONT FACE="Helvetica, Arial" SIZE="-1">
      <A HREF="JavaScript:PageCreator('Company1')">
        Company 1</A></FONT>
    </TD>
    <TD>
      <FONT FACE="Helvetica, Arial" SIZE="-1">
        <A HREF="JavaScript:PageCreator('Company2')">
          Company 2</A></FONT>
    </TD>
  </TR>
```

En la escritura de este *script* hemos utilizado algunos elementos de JavaScript nuevos. Vamos a verlos:

- El objeto **WINDOW**, del que hemos visto cómo abrir, cerrar y llenar ventanas.
- Rellenar y acceder a la información de la tabla según la posición numérica que la información ocupa y mediante las etiquetas definidas por el usuario.

Ahora tenemos un método claro y eficaz para tratar lo que de otra manera podría haber sido un trabajo bastante tedioso. Una de las grandes ventajas de JavaScript es que le ofrece el poder de proponer soluciones como ésta, que no sólo le ayudarán a solventar las cargas de trabajo, sino que realmente se trata de usos recientes de la tecnología. ¡Oh!, aquí viene el jefe con otro proyecto para usted. ¿Qué opina sobre comprobar lo siguiente a lo que se va a enfrentar?

Proyecto II: cómo utilizar JavaScript para comunicarse entre marcos

Seguro que hay una nueva cuestión que el jefe quiere que trate en el sitio web. Actualmente, en el marco inferior que soporta la navegación principal, los gráficos para la sección en la que se encuentra aparecen destacados (véase la Figura 5.2). Otro programador ha escrito el código JavaScript necesario para cuidar el cambio de las imágenes cuando se haga clic en ellas, pero el jefe necesita que usted se preocupe de otro problema.

Si el usuario utiliza los botones Adelante o Atrás en lugar de hacer clic en las imágenes, los gráficos no se actualizan. Por tanto, debemos configurar un *script* que haga que el marco de contenido se comunique con la navegación, de modo que cuando se cargue una página nueva en el marco de contenido se indicará al marco de navegación qué gráfico deberá estar destacado. Para este proyecto, vamos a necesitar un poco más de información acerca de cómo trabaja JavaScript con los marcos.

Desentrañando la jerarquía de marcos

Puede pensar en un marco como en un objeto sub-WINDOW; al igual que una ventana, contiene un objeto DOCUMENT, así como todos los objetos descendientes y propiedades que el código HTML puede crear. Sin embargo, si observa la jerarquía de JavaScript, verá que el objeto FRAME se sitúa por debajo del objeto WINDOW; de hecho, es una propiedad del objeto WINDOW.

Cada marco en una ventana puede tener URL separados. Esta capacidad puede resultar muy útil para el sitio Stitch, que utiliza tres marcos, uno para el contenido y dos para la navegación. Esto permite al usuario moverse en el marco del contenido manteniendo siempre la navegación en su lugar.

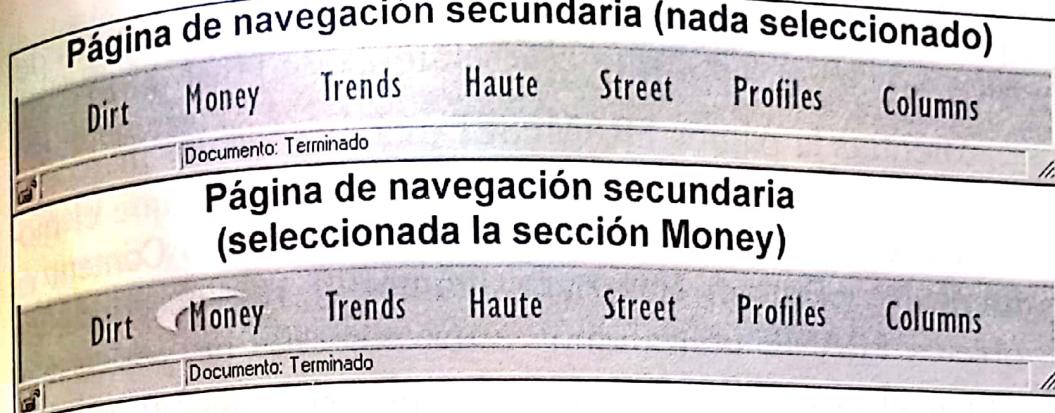


Figura 5.2
Barra de navegación inferior de una página secundaria de Stitch.

Cuando un documento HTML con un FRAMESET se carga en el navegador, el motor de JavaScript crea un array con todos los objetos FRAME que se crean. Esto es muy parecido a la forma en que JavaScript trabaja con las imágenes que encuentra en una página. La forma de hacer referencia a los objetos también es similar. Veamos el FRAMESET para el sitio Stitch y cómo podemos hacer referencia a los diversos objetos FRAME que se crean.

```
<FRAMESET ROWS="65,* ,50" FRAMEBORDER="0" BORDER="0"
FRAMESPACING="NO">

<FRAME NAME="Top_Nav_Bar" SRC="top_nav.html"
      MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="no"
      FRAMEBORDER="no" NORESIZE>

<FRAME NAME="Content" SRC="ad_index.html"
      MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="auto"
      FRAMEBORDER="no" NORESIZE>

<FRAME NAME="Bottom_Nav_Bar" SRC="bottom_nav.html"
      MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="no"
      FRAMEBORDER="no" NORESIZE>

</FRAMESET>
```

El código anterior es el código HTML que constituye el documento FRAMESET. Hay tres marcos existentes dentro del FRAMESET: Top_Nav_Bar, que mantiene la navegación común y el logotipo de cabecera; Content, que incluye todo el contenido de las secciones del sitio; y Bottom_Nav_Bar,

que mantiene la navegación de las secciones principales del sitio. La Figura 5.3 ilustra esta jerarquía de FRAMESET.

Mientras la página FRAMESET está cargada, los marcos se almacenan en el array FRAMES en el orden en que el motor los lee: Top_Nav_Bar es frames[0], el marco Content es frames[1] y Bottom_Nav_Bar es frames[2]. Para acceder a los objetos FRAME utilizaremos estos números de posición al llamar al array FRAMES. Por ejemplo, si quisieramos acceder al URL del marco Content, la sintaxis sería así:

frames[1].src

Sin embargo, esto sólo funciona si está solicitando información de la página que contiene el FRAMESET. Si estuviéramos en uno de los otros marcos, tendríamos que decirle al motor de JavaScript que primero volviera al nivel superior

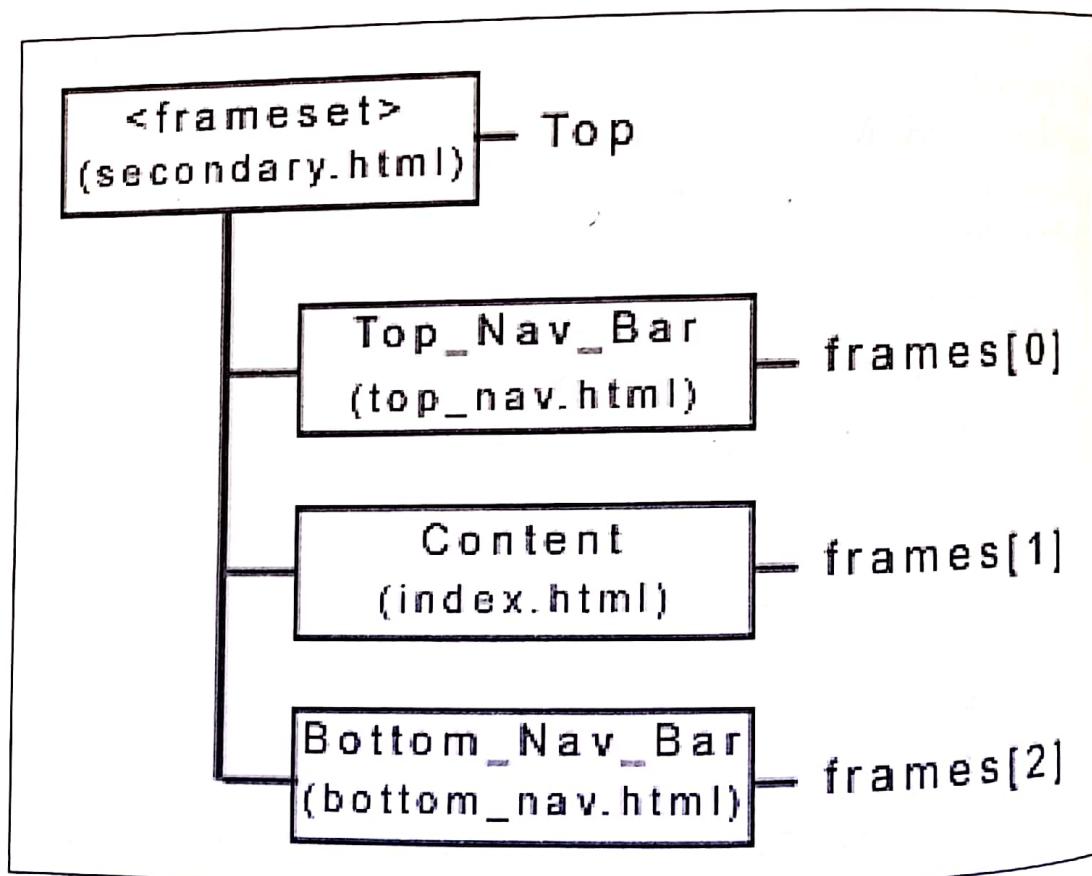


Figura 5.3
Nuestra jerarquía FRAME.

de la ventana y después fuera al marco Content. Hay dos formas de hacerlo. Puede usar esta línea:

```
top.frames[1].src
```

para indicarle al motor que vaya al nivel superior de la jerarquía y después vaya al segundo marco en el array FRAMES. También puede utilizar parent en lugar de top para conseguir el mismo efecto:

```
parent.frames[1].src
```

La diferencia es que top envía al motor a la cima de la jerarquía, mientras que parent sólo sube el puntero un nivel, al objeto que es el padre del marco desde el que se efectúa la llamada. Habrá veces en las que tropiece con marcos anidados; en otras palabras, el documento HTML que se llama en un marco contiene otro FRAMESET en lugar de código HTML. Esto añadirá otro grupo de capas a la jerarquía, de modo que para llegar a la cima de la jerarquía necesitará usar parent varias veces:

```
parent.parent.frames[1].src
```

Trabajar con marcos anidados en JavaScript puede llegar a ser bastante complejo. Afortunadamente, el sitio Stitch no contiene marcos anidados. Dicho esto, sigamos con el proyecto. Este proyecto se compone de dos secciones: primero, necesitaremos crear una función en Bottom_Nav_Bar que cambiará el gráfico destacado cuando la página Content cambie. Segundo, insertaremos manipuladores de eventos en todas las páginas que llenen el marco Content, que llamarán a la función que creamos en la primera parte del proyecto.

Creación de una función para comprobar en qué sección está

Antes de escribir la nueva función, será útil echar un vistazo a la función existente, entendiendo que podemos ser

capaces de utilizar esta función en el *script*. Esta función realiza dos tareas: primero, cuando se llama después de que el usuario haga clic en una de las imágenes de navegación, recorre todas las imágenes de navegación poniéndolas en Off. Una vez hecho esto, vuelve y mira para encontrar el gráfico en el que el usuario hizo clic devolviendo esa imagen al estado On. Observe la función en sí misma de manera que pueda obtener una idea de cómo se lleva a cabo.

```
function Highlighter(current)
{
    var test = 1;
    while (test < 8)
    {
        document.images[test].src = eval("Pic" + test +
            "Off.src");
        ++ test;
    }
    document.images[current].src = eval(current +
        "On.src");
    able = current;
}
```

En la primera línea de la función, la variable *test* se configura a 1; este valor se va a usar como pregunta en el bucle *while* que sigue. La siguiente línea es el inicio del bucle *while*; mientras *test* sea menor que 8, el bucle seguirá ejecutándose. Con *test* configurado a 1 cuando active la función, el bucle se ejecutará siete veces, una por cada imagen de navegación. Comenzamos con el valor de *test* a 1 en lugar de a 0, ya que la imagen de fondo para el marco toma la primera posición en el array **IMAGE** y necesitamos saltarla cuando vamos a desactivar las imágenes.

Dentro del bucle cambiaremos la fuente de la imagen localizada en la misma posición como el valor de *test* a la ver-

sión off del gráfico. Cuando el bucle while finalice, todos los objetos Navegation habrán vuelto a su estado Off predeterminado.

Ahora que todas las imágenes están en estado Off, es el momento de destacar la que el usuario ha marcado. La siguiente línea de la función hace esto, configurando el origen de la imagen correspondiente al valor de la variable current a la posición *rolled-over*.

La línea final de la función configura el valor de la variable able al nombre de la categoría en que se ha hecho clic, que está contenida en la variable current. La variable able se usa como pregunta en la función off(), una de las dos funciones que manipulan los *rollovers* de la página.

```
function off(pic)
{
    if(document.images)
    {
        if (pic != able)
        {
            document.images[pic].src= eval(pic +
                "Off.src");
        }
    }
}
```

Por tanto, si el gráfico de la sección en la que está actualmente no tiene el foco, no se desactivará. También usaremos esta variable en la función.

Ahora que hemos visto lo que ocurrirá en la página, es el momento de insertar la función. Comparada con la función existente, la nuestra debería ser pan comido. La función va a ser llamada cada vez que se cargue una nueva página HTML en el marco Content, pero sólo la queremos para

cambiar una imagen si la página que se carga pertenece a una sección diferente de la página anterior. Para hacer esto, pasaremos un valor a la función cuando ésta sea llamada; este valor será el nombre de la sección en que está la página. Entonces, compararemos ese valor con el valor de la variable `able`, que contiene el nombre de la sección en la que el usuario hizo clic por última vez. Sólo ejecutaremos el resto de la función si estos valores son diferentes; en otras palabras, si la sección ha cambiado. Dicho esto, comenzaremos con la función.

```
function SectionChecker(content)
{
    if (content != able)
    {
        ...
    }
}
```

En el código anterior, comenzamos la función y añadimos la sentencia `if` que comparará el valor de `content` con el de `able`. Ahora, sólo hay algo más que debemos añadir para finalizar la función, insertar el código que queremos ejecutar si los valores difieren.

```
function SectionChecker(content)
{
    if (content != able)
    {
        Highlighter(content);
    }
}
```

Debido a que la función existente ya contiene la función `Highlighter()`, que desactiva los siete gráficos y después activa aquél con el nombre que pasamos a la función, todo lo

que tenemos que hacer es llamar a la función y pasarle el valor que se le dio a la función por la página de contenido.

Como resultado, si la página que está cargada en el marco Content pasa el valor `dirt` a la función, entonces la función llamará a la función `Highlighter()` y pasará el mismo valor, lo que, en esencia, desactivará todas las imágenes de navegación excepto la imagen `dirt`. Ahora, todo lo que tenemos que hacer es insertar los manipuladores de eventos en todas las páginas que se sitúen en el marco Content.

Inserción del manipulador de eventos

Hasta ahora nos hemos basado en manipuladores de eventos que reaccionan a alguna interacción con el usuario: *rollover* de una imagen, hacer clic en un botón o cambiar algún campo de formulario. Sin embargo, para este proyecto, lo que necesitamos es un manipulador que se ejecute cuando el navegador cargue una página HTML. Bien, éste es nuestro día de suerte; el manipulador de eventos `onLoad()` hace justo eso. Todo lo que necesitamos es situar el manipulador de eventos `onLoad()` en la etiqueta `<BODY>` del documento HTML. Para nuestros propósitos, pondremos una de las páginas de Content para la sección `Dirt`.

```
<BODY onLoad=". . . ">
```

Ahora que hemos insertado el manipulador de eventos, vamos a introducir la llamada a la función `SectionChecker()`. Recuerde que debido a que la función `SectionChecker()` está localizada en otro marco, tendremos que especificar la ruta a la función cuando la llamemos.

```
<BODY onLoad="top.frames[2].SectionChecker('dirt')">
```

Lo crea o no, esto es todo; sólo necesitará poner este manipulador de eventos en todas las páginas HTML para el marco Content. El único elemento que variará es el valor

que va a pasar a la función `SectionChecker()`; este valor será el nombre de la sección en la que resida la página.

Sin embargo, no lo hemos hecho todavía. Uno de los programadores de la compañía estaba comprobando la nueva adición al sitio y encontró que podría ser un problema potencial con el *script*. Es posible que el manipulador de eventos `onLoad()` ejecute antes el marco `Bottom_Nav_Bar` que contiene la función que se carga. Si esto ocurre, se generará un error JavaScript.

Para evitar que ocurra, debemos asegurarnos de que el marco `Bottom_Nav_Bar` se carga antes de la llamada a la función `SectionChecker()` desde el marco. Por tanto, necesitamos añadir algo de código a la página `FRAMESET`, la página `Bottom_Nav_Bar` y todas las páginas `Content`. Primero, insertaremos el código necesario en la página `FRAMESET`. Sabemos que si el marco `Content` o el marco `Bottom_Nav_Bar` se cargan, la página `FRAMESET` en sí misma deberá cargarse, de manera que utilizaremos una variable situada en la página `FRAMESET` para ejecutar la pregunta.

```
<SCRIPT Language="JavaScript">  
  <!-- El código a partir de aquí será ignorado por los  
  ➔ navegadores más antiguos  
  var capable = 'no';  
  // Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Todo lo queharemos en el código anterior es crear una variable llamada `capable` y asignarle el valor predeterminado `no`. Cuando la página `FRAMESET` se cargue, se creará la variable `capable`. Después, situaremos el código que necesitamos en el marco `Bottom_Nav_Bar`. Necesitaremos insertar una línea de código que llamará a la variable `capable` en la página `FRAMESET` y le asignará el valor de `yes`; puede introducirla justo después de la función `SectionChecker()`.

```
function SectionChecker(content)
{
    if (content != able)
    {
        Highlighter(content);
    }
}
top.capable = 'yes';
// Dejamos de ocultar el código aquí ->
</SCRIPT>
```

De nuevo, como con el manipulador de eventos `onLoad()`, necesitamos decirle al navegador cómo llegar a la variable en la jerarquía. Cuando el marco `Bottom_Nav_Bar` se cargue, cambiará la variable `capable` para que contenga el valor `yes`.

La última parte de código que necesitamos insertar está en las páginas `Content`. Hagamos el cambio a la misma sección de página `Dirt` que ya pusimos en el manipulador de eventos `onLoad()`. Lo primero que necesitamos hacer es cambiar el manipulador de eventos `onLoad()`.

```
<BODY BGCOLOR="White" onLoad="Checker()">
```

En lugar de llamar al marco `SectionChecker()` directamente como hicimos antes, vamos a crear una nueva función en la página `Content` en sí misma que llamará a dicha función. Esta nueva función, `Checker()`, es la función que vamos a llamar en el manipulador de eventos `onLoad()`. Una vez que creemos la función `Checker()`, estará hecho.

```
<SCRIPT Language="JavaScript">
<!-- El código a partir de aquí será ignorado por los
    navegadores más antiguos
function Checker()
{</pre>
```

```
if (top.capable='yes')  
{  
    top.frames[2].SectionChecker('dirt');  
}  
}  
  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Esta función es bastante sencilla. Lo primero que ocurre cuando se llama es que utilizamos una sentencia `if` para comprobar el valor de la variable `capable` en la página FRAME-SET. Si el marco `Bottom_Nav_Bar` se ha cargado, el valor de `capable` habrá cambiado a `yes`, de modo que podemos llamar a la función `SectionChecker()`. Sin embargo, si el marco `Bottom_Nav_Bar` no se ha cargado, no ocurrirá nada.

Todo lo que debe hacer es insertar el manipulador de eventos `onLoad()` y la función `Checker()` en las demás páginas localizadas en el marco Content. Después, puede usar los botones Atrás y Adelante para su contenido y siempre sabrá en qué sección está.

ANÁLISIS DEL *SCRIPT*

Echemos un vistazo al *script* y cómo lo construimos.

1. Primero, incorporamos la funcionalidad de un *script* existente en este proyecto.

```
function Highlighter(current)  
{  
  
    var test = 1;  
    while (test < 8)  
    {  
  
        document.images[test].src = eval("Pic" + test +  
        "Off.src");  
        ++ test;  
    }  
}
```

```
        }
        document.images[current].src = eval(current +
"on.src");
        able = current;
    }
}
```

2. Despu s, añadimos la funci n **SectionChecker()** a la p gina localizada en el marco **Bottom_Nav_Bar**. Esta llamar  a la funci n **Highlighter()** y le pasar  el nombre de la secci n a la que pertenece la p gina cargada en el marco **Content**.

```
function SectionChecker(content)
{
    if (content != able)
    {
        Highlighter(content);
    }
}
```

3. Despu s, insertamos el manipulador de eventos **onLoad()** en todas las p ginas HTML que se mostrar n en el marco **Content**. Insert mos en el manipulador de eventos una llamada a la funci n **SectionChecker()** y le pasamos el nombre de la secci n a la que pertenece la p gina.

```
<BODY onLoad="top.frames[2].SectionChecker('dirt')">
```

Una vez que los manipuladores de eventos estaban insertados, el *script* era funcional; sin embargo, para hacerlo m s fiable decidimosadir un toque final que nos evitara errores si el marco **Bottom_Nav_Bar** no se hubiera cargado antes que la funci n **SectionChecker()**.

1. Primero, añadimos una variable a la p gina HTML que conten a el **FRAMESET**. Usamos esta variable para comprobar si el marco **Bottom_Nav_Bar** se ha cargado o no.

```
<SCRIPT Language="JavaScript">
```

```
<!-- El código a partir de aquí será ignorado por los  
→ navegadores más antiguos  
  
var capable = 'no';  
  
// Dejamos de ocultar el código aquí -->  
  
</SCRIPT>
```

2. Después, añadimos una línea de código a la página HTML en el marco Bottom_Nav_Bar que cambiará el valor de la variable en la página FRAMESET para decírnos que el navegador ha cargado la página Navigation.

```
function SectionChecker(content)  
{  
    if (content != able)  
    {  
        Highlighter(content);  
    }  
  
    top.capable = 'yes';  
  
    // Dejamos de ocultar el código aquí -->  
}</SCRIPT>
```

3. Finalmente, añadimos una función a todas las páginas HTML que van en el marco Content y modificamos el manipulador de eventos `onLoad()` para llamar a esta nueva función en lugar de llamar a la función `SectionChecker()` directamente.

```
<SCRIPT Language="JavaScript">  
  
<!-- El código a partir de aquí será ignorado por los  
→ navegadores más antiguos  
  
function Checker()  
{  
    if (top.capable='yes')  
    {  
        top.frames[2].SectionChecker('dirt');  
    }  
}</SCRIPT>
```

```
    }

}

// Dejamos de ocultar el código aquí -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="White" onLoad="Checker()">
```

Resumen

Trabajar con ventanas y marcos puede incrementar la funcionalidad de un sitio web (cuando se usan con JavaScript, su imaginación es el límite de lo que puede realizar). En este capítulo sólo hemos expuesto lo que puede hacer con estos elementos. El concepto de los marcos y el entendimiento de su jerarquía no resulta fácil de dominar, pero una vez que tenga los fundamentos abrirá un amplio rango de nuevas aplicaciones.

Proyectos avanzados

1. Utilice la técnica empleada en el primer proyecto de este capítulo en un sitio con marcos anidados, donde tendrá dos grupos de imágenes de navegación que necesitan ser seguidas, uno para la navegación principal y otro para la navegación secundaria.
2. Cree sitios basados en marcos con un grupo de marcos anidados donde pueda ocultar el marco de navegación cambiando la propiedad de localización de marco que mantiene los marcos internos.

JavaScript avanzado

6

En este capítulo

- Proyecto I: cómo crear submenús para las páginas secundarias de Stitch.
- Resumen.
- Proyectos avanzados.

Ahora que ya tiene los conocimientos básicos, va siendo hora de que hagamos algo más complicado.

Proyecto I: cómo crear submenús para las páginas secundarias de Stitch

Los estudios realizados nos muestran que los usuarios tienden a perderse en la mayoría de los sitios web (se pasan la mayoría del tiempo intentando encontrar información). Dichos estudios también demuestran que la mayoría de usuarios visitan la Web para encontrar una información específica. Se preguntará, ¿quién lee esos estudios? Pues el jefe, seguro. De hecho, quiere que se presente con algunos conceptos que mejoren la navegación en la página principal para Stitch. Así que va a tener que sufrir bastantes dolores

de cabeza entrevistándose con otros programadores. Cuando sea su turno para proponer alguna idea, sugiera un tema de submenús gráficos, donde los usuarios puedan previsualizar qué hay en la sección, antes de acceder a ella. Después de que la haya propuesto, verá cómo se iluminan los ojos del jefe. Adivine cuál va a ser el próximo proyecto. Es cierto. Una nueva página principal para Stitch con un sistema de submenús mejorado mediante JavaScript.

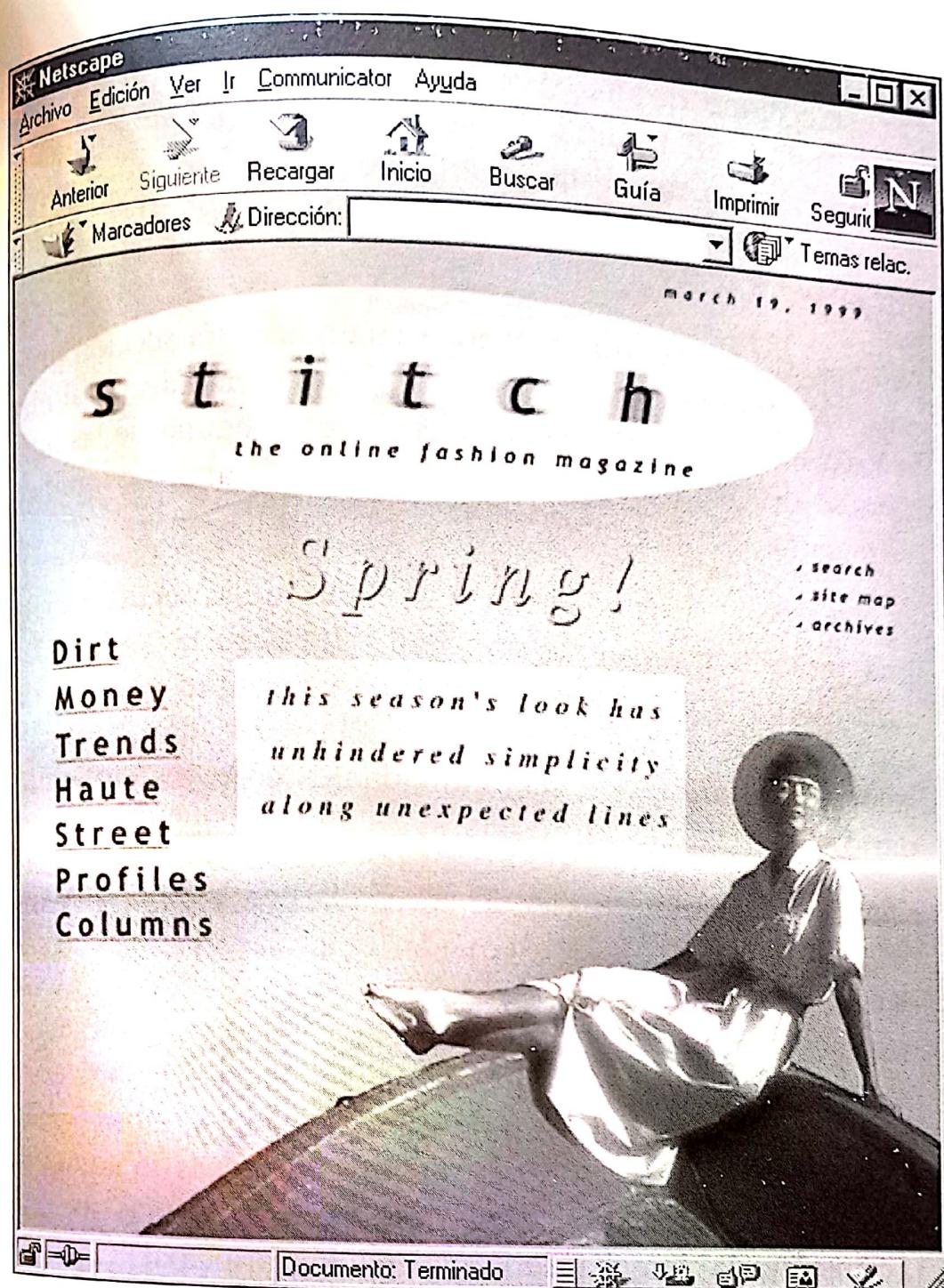
Especificaciones de funcionalidad

Ya que esto es un poco complicado, vamos a aclarar qué es exactamente lo que queremos que ocurra antes de empezar con los códigos. Es tentador saltar este paso y pasar directamente al código, pero tiene que controlarse. Piense en el proyecto antes de comenzar, para que ahorre tiempo y obtenga siempre los mejores resultados en el producto final.

La Figura 6.1 es una imagen de la página principal de Stitch. Ya tiene un bonito diseño, y un sistema de submenús debería funcionar bien con dicho diseño.

Actualmente, tenemos el logotipo en la parte superior, con el título para la temporada de moda de primavera y un pequeño menú de recursos debajo y en el lado derecho. Las categorías de navegación principal están situadas a la izquierda. Un pequeño anuncio gráfico se encuentra en mitad de la página. Finalmente, hay una fotografía de una mujer sentada en un barco, en la parte inferior derecha de la página.

Lo que tenemos que hacer es permitir que el usuario seleccione una de las opciones de navegación principales, tener otro menú contextual y reemplazar el anuncio gráfico. Por ejemplo, si un usuario selecciona **Haute**, necesitamos que aparezca una lista de los artículos de la sección “Haute”

**Figura 6.1**

Página principal actual de Stitch.

donde está el anuncio. Después, el usuario debería poder navegar directamente hasta el artículo, evitando la página principal de "Haute", haciendo clic en uno de los artículos que se encontrarían en los subniveles.

Debería aparecer un nuevo menú cada vez que el usuario seleccione de una sección principal del sitio. Por ejemplo, cuando el usuario seleccione **Street**, debería aparecer un menú con una lista de artículos en la sección “Street”. Cuando seleccione **Profiles** aparecería una lista de artículos en la sección “Profiles”.

Cuando el usuario seleccione cualquier otra sección principal, el menú de la sección secundaria reemplazaría al menú existente. Cuando no se seleccionen ninguna de las imágenes de la navegación principal, el anuncio gráfico regresaría a su posición.

Y hay más. Una vez que aparezca el menú secundario, la imagen de la navegación principal seleccionada por el usuario se debería resaltar de algún modo para que éste supiera qué menú secundario está viendo. El usuario debería poder seleccionar también temas individuales en el menú secundario, que se deberían resaltar.

Vamos a comenzar viendo la realidad de la página. Al introducir una navegación secundaria, no sólo vamos a cambiar la imagen de la categoría que selecciona el usuario, sino que también necesitaremos una imagen nueva que contenga las categorías secundarias que aparecerán (véase la Figura 6.2). ¿Adónde va a ir esta imagen? Afortunadamente, ésta es la parte fácil. Como hemos mencionado anteriormente, reemplazaremos el anuncio de texto gráfico que está situado a la derecha de la navegación principal.

Estos son los diferentes objetivos que tenemos que lograr con ayuda del *script*:

1. Identificar la categoría de la navegación primaria (por ejemplo, **Dirt**, **Street**, etc.) por la que se está moviendo el usuario.
2. Resaltar dicha categoría con un *rollover*.

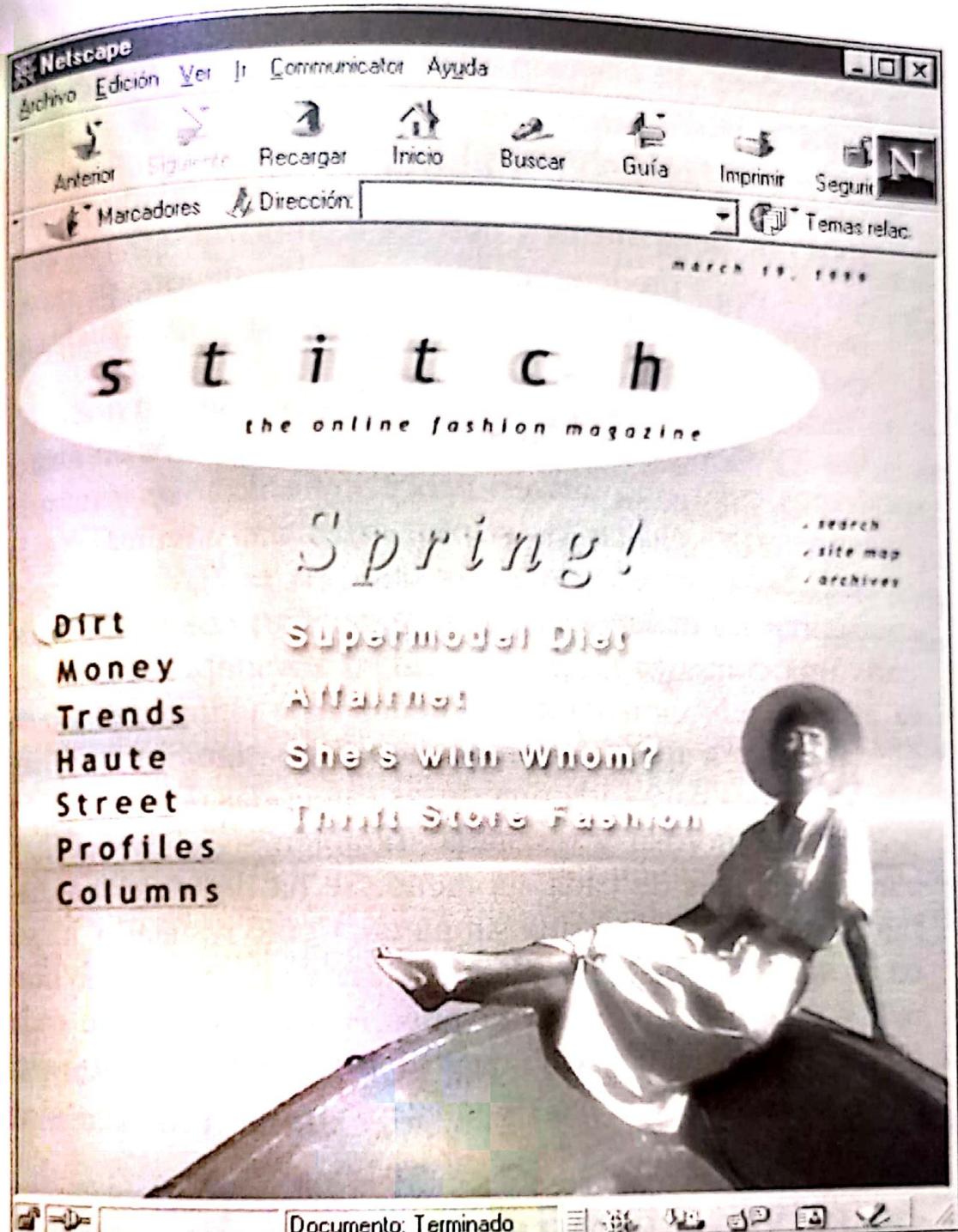


Figura 6.2

La página principal muestra la navegación secundaria para la sección Dirt.

3. Reemplazar el anuncio de texto con el submenú de navegación secundario apropiado.

- i. Cuando el usuario seleccione los temas de la navegación secundaria, la elección individual que se seleccione se deberá resaltar.
- j. Cuando el usuario abandone el menú secundario, queremos que desaparezca y que sea reemplazado por el anuncio de texto predeterminado, y que la categoría de navegación principal deje de estar resaltada y que vuelva al estado predeterminado.

Existen dos maneras de dividir las imágenes de un menú: podemos crear una imagen sencilla y utilizar un mapa de imágenes para ajustar los puntos calientes, y reemplazar la imagen cada vez que se seleccione una categoría; o podemos cortar las imágenes de la navegación primaria para que cada una contenga su propio gráfico y reemplazar después la imagen seleccionada actualmente. Este último método se ajusta mejor a nuestros propósitos. Esto significa que hay dos versiones para cada una de las categorías de navegación principales: normal y resaltada. También significa que hay varias versiones de cada submenú. Se ha decidido que habrá cuatro enlaces en cada submenú. Como resultado, habrá cinco versiones de la imagen del submenú: una imagen normal y cuatro con varias opciones resaltadas. Ya que cada imagen del submenú tiene cuatro opciones diferentes, necesitaremos crear un mapa de imagen.

<PLUG TYPE="SHAMELESS">

Esta clase de técnica de navegación/submenús es mucho más fácil en DHTML. Si quiere aprender más sobre ella, consulte la **Guía esencial CSS y DHTML**. Asumimos que sabe algo de HTML y JavaScript y sería buena idea que leyera un poco después de acabar este libro. En serio.

</PLUG>

Para el sitio de Stitch hay siete imágenes diferentes de navegación principal, lo que hace una suma de catorce imágenes diferentes (una normal y una resaltada para cada una). En la parte superior, hemos decidido limitar los submenús a cuatro opciones, por lo que cada submenú necesita cinco imágenes independientes. Esto hace un subtotal de treinta y cinco imágenes de submenús, y un total de cuarenta y nueve imágenes.

Son muchas imágenes que manipular. Puede construir varias sentencias `if` para seguirles la pista, o puede construir un sistema que investigue toda la información que necesita. Para hacerle la vida más fácil, vamos a elegir la última. Y, ya que ésta es una revista con un contenido cambiante, sería agradable construir algún código que sea fácil de actualizar con enlaces nuevos o con nuevas secciones.

También debería observar que, ya que este método va a involucrar a muchas imágenes, sería prudente asegurarse de que dichas imágenes tienen un tamaño de archivo muy pequeño, o correrá el riesgo de perder un tiempo excesivo en la descarga. Vamos a tener que decir al diseñador que nos haga unos diseños minimalistas.

Creación y población de los arrays con los datos de almacenamiento

El primer paso del proyecto va a ser la creación de dos variables globales: `whichMenu` para vigilar la categoría de navegación principal que ha seleccionado el usuario, además de qué submenú secundario se debe mostrar, y `overImage`, que nos ayudará a asegurar que desactivamos las imágenes sólo cuando sea necesario.

Para mantener la pista de todas las imágenes, tenemos que crear un array multidimensional, al que llamaremos `section`. Hasta ahora, hemos expuesto sólo los arrays unidimensionales.

mensionales. Los multidimensionales trabajan de una forma muy similar, pero nos ofrecen mayores capacidades. Si cree que un array dimensional es como un armario archivador, y que cada una de las entradas del array es como un cajón que guarda un pedazo de información, entonces, un array multidimensional tendría varias carpetas de archivos en cada uno de los cajones del armario. Esto haría que cada cajón del armario archivador principal fuese un armario archivador en sí mismo.

En el array `section`, la primera dimensión serán las siete categorías principales, que mantendrán a su vez a la segunda dimensión, la cual almacenará todas las elecciones individuales para el submenú de las categorías. Vamos a construirlo. Primero definiremos las dos variables globales:

```
<SCRIPT Language="JavaScript ">  
  <!-- El código a partir de aquí será ignorado por los  
      → navegadores más antiguos  
  
  var whichMenu=" ";  
  var overImage = 'no'  
  
  ...  
  
  // Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Hemos asignado a las dos variables valores predeterminados que harán que tengan más sentido mientras trabajamos con el *script*. Por ahora, vamos a dirigirnos a la creación de un array.

```
<SCRIPT Language="JavaScript ">  
  <!-- El código a partir de aquí será ignorado por los  
      → navegadores más antiguos  
  
  var whichMenu=" ";  
  var overImage = 'no'  
  
  section = new Array(4);  
  
  section['dirt'] = new Array (4)  
    section['dirt'][0] = 'supermodel'
```

```
section['dirt'][1] = 'affairnet'
section['dirt'][2] = 'moss'
section['dirt'][3] = 'married'

section['money'] = new Array (4)
section['money'][0] = 'worldclique'
section['money'][1] = 'germany'
section['money'][2] = 'mergers'
section['money'][3] = 'nmearnings'

section['trends'] = new Array (4)
section['trends'][0] = 'tshirts'
section['trends'][1] = 'sweatshirts'
section['trends'][2] = 'hats'
section['trends'][3] = 'mugs'

section['haute'] = new Array (4)
section['haute'][0] = 'paris'
section['haute'][1] = 'gaudy'
section['haute'][2] = 'taffeta'
section['haute'][3] = 'royalty'

section['street'] = new Array (4)
section['street'][0] = 'angry'
section['street'][1] = 'prison'
section['street'][2] = 'skater'
section['street'][3] = 'hip_hop'

section['profiles'] = new Array (4)
section['profiles'][0] = 'moxon'
section['profiles'][1] = 'west'
section['profiles'][2] = 'trinidadaddy'
section['profiles'][3] = 'spurn'

section['columns'] = new Array (4)
section['columns'][0] = 'column1'
section['columns'][1] = 'column2'
section['columns'][2] = 'column3'
section['columns'][3] = 'column4'

...
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Vamos a echar un vistazo detenidamente a lo que ha sucedido en el código anterior. Ésta es la primera línea que hemos añadido:

```
section = new Array(4);
```

Esta línea debería serle muy familiar (hemos creado arrays para la sección de anunciantes del sitio Stitch empleando el mismo método). Esta línea le dice al motor de ejecución de JavaScript que cree un array que se llama `section` y que le asigne una longitud de 4. En la línea siguiente nos adentraremos ya en un territorio nuevo.

```
section['dirt'] = new Array (4)
```

Esta línea crea un array nuevo y lo inserta en el array `section` en la posición `dirt`. Ya hemos dado el primer paso para crear el array multidimensional. Después rellenamos el array `dirt`.

```
section['dirt'][0] = 'supermodel'  
section['dirt'][1] = 'affairnet'  
section['dirt'][2] = 'moss'  
section['dirt'][3] = 'married'
```

Una vez configurado el array `dirt`, creamos el resto del código y rellenamos los arrays que almacenarán la información para las otras seis categorías.

Veamos los nombres que hemos asignado a la información almacenada en los arrays `CATEGORY`. Dichos nombres tienen un significado, y éste será importante más adelante para el *script*. Hasta ahora, siempre hemos creado nuevos objetos `IMAGE` para almacenar las imágenes que utilizaremos para los *rollovers*. Para este proyecto vamos a emplear otro método. Cuando reemplazemos imágenes en este *script*, sólo vamos a ajustar la propiedad `source` de la imagen inicial a una ruta de acceso nueva y los nombres que hemos almacenado en los arrays van a ayudarnos a conseguirlo. Por ejemplo, hay un artículo en la sección `Dirt` titulado "Supermodel Diet", así que cuando el usuario tenga dicha sección resaltada y el submenú desplegado, una de las opciones del submenú será "Supermodel Diet". Ahora, si el

usuario se mueve por encima de esta opción con el cursor, debemos intercambiar el submenú predeterminado por otro que tenga resaltada la opción Supermodel. Llamando a este gráfico supermodel.gif, podemos emplear la información almacenada en los arrays para crear sobre la marcha el nombre del archivo gif, mientras cambiamos la propiedad source del submenú predeterminado. Esto puede parecer un poco complejo en principio, pero pronto tendrá sentido.

Creación de las funciones para ejecutar el sistema de submenús

Ahora que ya hemos creado y poblado los arrays, tenemos que crear varias funciones para ejecutar el sistema de submenús. Vamos a comenzar con la función a la que se llamará cuando seleccionemos una de las imágenes de navegación principales.

```
function mainNavOn (sectionName)
{
    overImage='yes';
    // asegurarse de que el usuario se mueve por encima
    // de una imagen de navegación principal
    if (whichMenu != " ")
    {
        ...
    }
    ...
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Hay muchos modos de resaltar los gráficos de la página principal; queremos saber cuándo el usuario está sobre una imagen. La primera línea de código de la nueva función nos

ayudará a vigilar esto último. Ya que a esta función se la llama sólo cuando el usuario está sobre una de las imágenes de navegación principales, vamos a ajustar la variable `overImage` a un valor de `yes`.

La línea siguiente del código nos ayudará a desactivar la imagen que el usuario ha dejado. Para hacerlo, utilizaremos la variable `whichImage` que hemos declarado al comienzo del *script*. Cuando tengamos la función un poco más avanzada, tendremos que reajustar esta variable para que contenga el nombre de la sección que el usuario ha seleccionado. Al igual que al principio de la función, la variable `whichImage` va a contener el nombre de la última categoría que ha seleccionado el usuario. En el caso de que no hubiese seleccionado nada todavía, mantendrá el valor que le hemos asignado al principio del *script*. La siguiente línea de código del *script* es una instrucción que le dice al motor de ejecución de JavaScript que ejecute la siguiente parte de código, mientras el valor de `whichImage` no contenga el valor predeterminado.

Vamos a añadir el código que va a residir en el interior de la sentencia `if` y veamos lo que hace.

```
function mainNavOn (sectionName)
{
    overImage='yes';

    // asegurarse de que el usuario se mueve por encima
    // de una imagen de navegación principal
    if (whichMenu != " ")
    {
        // dejar de resaltar el nombre de sección antiguo
        eval ("document." + whichMenu + ".src =
            'images/nav/" + whichMenu + ".gif');");
    }
    ...
}
```

// Dejamos de ocultar el código aquí -->
</SCRIPT>

Esta línea de código es el primer ejemplo real de cómo manipularemos las sustituciones de imágenes en este *script*. Utilizar este método necesita dosis de disciplina, por lo que tenga mucho cuidado y determine si lo que va a hacer es beneficioso. En relación al argumento, supongamos que el usuario selecciona la categoría *Dirt*. Si se da este caso, la variable *whichImage* contendría el valor *dirt*. Veamos cómo se lee la línea nueva de código con este valor.

document.dirt.src = 'images/nav/dirt.gif';

La línea anterior intentará cambiar la propiedad *source* del objeto **IMAGE** *dirt* por *dirt.gif*, ubicado en el directorio *images/nav/*. Para hacer que este método funcione, necesitaremos que todas las versiones predeterminadas sin resaltar de las categorías principales sean nombradas empleando la sintaxis siguiente:

"Nombre de la categoría" + .gif

Las versiones resaltadas de los gráficos también necesitarán usar la siguiente sintaxis en los nombres:

"Nombre de la categoría" + _over.gif

Ahora debería comprender mejor el nuevo método para reemplazar imágenes. Este método es muy bueno si conoce la convención de nombres que se usa para las imágenes. Si no lo conoce, no se puede garantizar que las nombre adecuadamente, por lo que será mejor que continúe empleando el método que ha estado utilizando.

Cuando la imagen antigua ya no esté resaltada, necesitamos actualizar la variable *whichMenu* para mantener el nombre de la categoría sobre la que está el usuario, ahora que podemos resaltar la imagen.

function mainNavOn (sectionName)

```
{  
    overImage='yes';  
  
    // asegurarse de que el usuario se mueve por encima  
    // de una imagen de navegación principal  
    if (whichMenu != " ")  
  
    {  
  
        // dejar de resaltar el nombre de sección antiguo  
        eval ("document." + whichMenu + ".src =  
            'images/nav/" + whichMenu + ".gif'");  
  
    }  
  
    // establecer variable al rollover nuevo  
    whichMenu = sectionName;  
  
    ...  
}  
  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

En esta nueva línea hemos ajustado `whichImage` para que almacene el valor de `sectionName`, que es el nombre de la sección elegida por el usuario. Una vez hayamos hecho esto, añadiremos un nuevo código para cambiar la imagen que está seleccionada actualmente por la versión resaltada.

```
function mainNavOn (sectionName)  
{  
    overImage='yes';  
  
    // asegurarse de que el usuario se mueve por encima  
    // de una imagen de navegación principal  
    if (whichMenu != " ")  
  
    {  
  
        eval ("document." + whichMenu + ".src =  
            'images/nav/" + whichMenu + ".gif'");  
  
    }  
  
    // establecer variable al rollover nuevo  
    whichMenu = sectionName;
```

```
// resaltar la imagen nueva  
eval("document." + whichMenu + ".src = 'images/  
nav/" + whichMenu + "_over.gif'");  
  
...  
  
}  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Esta línea nueva debería parecerse mucho a la que usamos para desactivar la imagen seleccionada previamente. La única diferencia es que hemos añadido _over al nombre del gif. Como hemos seguido la convención de nombres, la imagen por la que el usuario se mueve se debería resaltar.

Finalmente, el último paso en esta función será colocar la imagen de submenú adecuada. Cuando comenzamos a colocar el submenú, no se resaltará ninguna de las categorías secundarias, por lo que llamaremos a la versión predeterminada del gráfico.

```
function mainNavOn (sectionName)  
{  
overImage='yes';  
// asegurarse de que el usuario se mueve por encima  
// de una imagen de navegación principal  
if (whichMenu != " ")  
{  
    eval ("document." + whichMenu + ".src =  
    'images/nav/" + whichMenu + ".gif'");  
}  
  
// establecer variable al rollover nuevo  
whichMenu = sectionName;  
// resaltar la imagen nueva  
eval("document." + whichMenu + ".src = 'images/nav/  
" + whichMenu + "_over.gif'");  
// hacer aparecer imagen de navegación secundaria  
eval("document_submenu.src = 'images/nav/" + whichMenu  
+ "_sec_blank.gif'");
```

```
    }  
    // Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Todas las imágenes del submenú deben seguir la siguiente sintaxis de nominación:

“Nombre de la categoría” + sec_blank.gif

Trataremos la sintaxis de la asignación de nombre para las versiones resaltadas de las imágenes del submenú en la función submenuOn(), que se cuida de resaltar la categoría del submenú adecuada.

De nuevo, el primer paso es establecer el valor de la variable overImage a yes, para que el *script* sepa que el usuario se encuentra sobre una imagen.

```
function submenuOn (sectionNum)  
{  
    overImage='yes';  
    ...  
}  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Cuando alguien se mueva por encima del área que es un mapa de imágenes de los submenús, debemos asegurarnos de que una de las imágenes de navegación principales ya ha sido seleccionada y que un submenú está visible antes de empezar a resaltar los elementos individuales del submenú. Si el usuario selecciona el anuncio de texto, no queremos que ninguno de los submenús se despliegue. Podemos determinar si un submenú está visible consultando el valor de la variable whichMenu. Si contiene el nombre de una sección en lugar del valor en blanco predeterminado, sabremos que una de las categorías principales ha sido seleccionada, y se mostrará un submenú secundario. Lo probaremos em-

pleando una sentencia if como la que está al principio de la primera función.

```
function submenuOn (sectionNum)
```

```
{ overImage='yes' ;
```

```
if (whichMenu != " ")
```

```
{
```

```
...
```

```
}
```

```
}
```

// Dejamos de ocultar el código aquí -->

```
</SCRIPT>
```

En la sentencia if necesitamos determinar qué categoría secundaria ha seleccionado el usuario en la imagen del submenú. Aquí es donde el array multidimensional comienza a ser conveniente.

```
function submenuOn (sectionNum)
```

```
{
```

```
overImage='yes' ;
```

```
if (whichMenu != " ")
```

```
{
```

```
// determina el nombre de subsección
```

```
// sobre la que se está moviendo el usuario
```

```
subSection = section[whichMenu][sectionNum];
```

```
...
```

```
}
```

```
}
```

// Dejamos de ocultar el código aquí -->

```
</SCRIPT>
```

En el código anterior hemos usado whichMenu para apuntar a la ubicación en el primer nivel del array que al-

macena el array anidado, el cual contiene la información para las categorías secundarias de la categoría principal actualmente seleccionada. sectionNum contiene un valor numérico que coincide con la posición de la categoría secundaria que ha seleccionado el usuario.

Si el usuario seleccionase la categoría secundaria Supermodel Diet de la sección Dirt, el lado derecho de la ecuación quedaría así:

```
section['dirt'][0]
```

Ésta, a su vez, llamaría a esa posición del array y devolvería el valor supermodel, que se almacenaría entonces en la variable subSection.

Ahora que tenemos la categoría de submenú secundaria almacenada en la variable, el siguiente paso es reemplazar el gráfico del submenú con el que tiene resaltada la categoría apropiada.

```
function submenuOn (sectionNum)
{
    overImage='yes';
    if (whichMenu != " ")
    {
        // determina el nombre de subsección
        // sobre la que se está moviendo el usuario
        subSection = section[whichMenu][sectionNum];

        // intercambia la imagen adecuada
        eval("document.submenu.src = 'images/nav/" +
        whichMenu + "_" + subSection + "_over.gif'");
    }
}

// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Al igual que con todas las sustituciones de imágenes anteriores, emplearemos una sintaxis de nombre específico. Para las categorías secundarias resaltadas, ésta es la sintaxis: "Nombre de la categoría principal" + "nombre de la categoría secundaria" + "_over.gif"

Ahora tenemos dos funciones que activarán varias imágenes. A continuación necesitamos crear las funciones que desactiven las imágenes resaltadas cuando el usuario pasa por encima de ellas. Al igual que con los *rollovers* de la página secundaria del Capítulo 2, tendremos que ser un poco audaces. No queremos que una imagen de navegación principal se desactive sólo porque hemos seleccionado el submenú, así que vamos a emplear dos funciones diferentes. Vamos con la primera.

```
function mapOff ()  
{  
    if (whichMenu != " ")  
    {  
        ...  
    }  
}  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

De nuevo, lo primero que tenemos que hacer es asegurarnos de que hasta que esta función se solicite, el usuario estuvo realmente sobre un gráfico. Así que crearemos una sentencia *if* para verificar el valor de *whichMenu*. Una vez determinado que se puede seguir adelante con el procedimiento, el siguiente paso será cambiar el valor de la variable *overImage* a *no*, mientras el usuario tiene seleccionada una imagen, para lo que se llamará a la función actual.

```
function mapOff ()  
{
```

```
if (whichMenu != " ")  
{  
    overImage = 'no';  
    ...  
}  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

El paso final es colocarla en el método `setTimeout`, que esperará un par de segundos, y llamar entonces a la segunda función que necesitaremos para desactivar las imágenes resaltadas.

```
function mapOff ()  
{  
    if (whichMenu != " ")  
    {  
        overImage = 'no';  
        setTimeout('mapOff2()',1000);  
    }  
}  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

Esta nueva línea le indica al motor que espere mil milisegundos (1 segundo) y llama a la segunda función `off, mapOff2()`. Este intermedio le da al usuario la oportunidad de seleccionar otro gráfico antes de que resalte las imágenes. Si el usuario se mueve entre las categorías secundarias del submenú, no necesitaremos tener la imagen de navegación principal mostrándose y ocultándose. Esto se va a solucionar en la segunda función `off`, así que vamos con ella.

Dondequier que el usuario se mueva por encima de una imagen, estableceremos el valor de la variable `overImage` a

yes. Utilizaremos esta variable para comprobar si tenemos que desactivar la imagen resaltada. Vamos a ver el primer paso de la función mapOff2().

```
function mapOff2()
{
    if ((overImage == 'no') && (whichMenu != " "))
    {
        ...
    }
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Antes de seguir adelante y desactivar cualquiera de las imágenes, nos aseguraremos de que el usuario no ha seleccionado otra imagen en el momento de llamar a esta nueva función. Hemos creado una sentencia if para vigilar esta posibilidad: si la variable overImage mantiene el valor no y la variable whichMenu no es el valor predeterminado, podemos desactivar cualquier gráfico que esté resaltado. Primero, desactivaremos el gráfico del submenú y lo devolveremos al anuncio de texto predeterminado.

```
function mapOff2()
{
    if ((overImage == 'no') && (whichMenu != " "))
    {
        ...
        // recupera la imagen del artículo
        document submenu.src = "images/nav/sec_blank.gif";
        ...
    }
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Esta nueva línea emplea el mismo método que el utilizado para el resto del *script*. Ahora, el submenú se debería desactivar con el anuncio de texto original de nuevo en su lugar. Ahora desactivaremos la imagen de navegación principal resaltada actualmente.

```
function mapOff2()
{
    if ((overImage == 'no') && (whichMenu != " "))
    {
        // recupera la imagen del artículo
        document submenu.src = "images/nav/sec_blank.gif";

        // cambiar la imagen de nav. principal
        // por la versión no resaltada
        eval ("document." + whichMenu + ".src =
'images/nav/" + whichMenu + ".gif'");
        ...
    }
    // Dejamos de ocultar el código aquí -->
}
</SCRIPT>
```

El paso final de esta función es restablecer la variable *whichMenu* al valor predeterminado.

```
function mapOff2()
{
    if ((overImage == 'no') && (whichMenu != " "))
    {
        // recupera la imagen del artículo
        document submenu.src =
"images/nav/ sec_blank.gif";

        // cambiar la imagen de nav. principal
        // por la versión no resaltada
        eval ("document." + whichMenu + ".src =
'images/nav/" + whichMenu + ".gif');");
    }
}
```

```
whichMenu = " ";
}

}

// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

Ya hemos acabado la segunda función off. Ahora necesitamos crear una función de navegación que envíe al usuario al URL correcto cuando haga clic en las categorías secundarias del submenú. Ya que sólo habrá un mapa de imágenes que controle los siete submenús, crearemos una función que determine qué submenú está desplegado y seleccione después el URL adecuado.

```
function goThere (sectionNum)
{
    if (whichMenu)
    {
        ...
    }
}

// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

De nuevo, y como en el resto de funciones, lo primero que tenemos que hacer es verificar y asegurarnos de que se ha seleccionado una categoría (no podemos enviar al usuario a cualquier sitio si accidentalmente hace clic en el anuncio de texto). Una vez que nos hayamos asegurado de que hay una categoría seleccionada, podemos continuar y averiguar dónde enviar al usuario.

```
function goThere (sectionNum)
{
    if (whichMenu)
```

```
{  
    subSection = section[whichMenu][sectionNum];  
    location.href = whichMenu + "/" + subSection +  
    ".html";  
  
}  
  
}  
  
// Dejamos de ocultar el código aquí -->  
</SCRIPT>
```

La primera nueva línea de código se inserta en el array, selecciona el nombre de la página del artículo y lo coloca en la variable `subSection`. Después, la línea siguiente utiliza esta variable, junto con el valor de `whichMenu`, para establecer un nuevo valor para la propiedad ubicación del objeto `DOCUMENT`.

Al igual que con las imágenes, confiamos en el hecho de que el contenido de cada categoría está en un directorio que tiene el mismo nombre que la sección. Por ejemplo, el artículo “Supermodel Diets” en la sección `Dirt` está ubicado en `dirt/supermodel.html`. De nuevo, esto nos ahorra tiempo al codificar, pero deberá asegurarse de que domina perfectamente la convención de nombres de archivos.

Sólo un paso más antes de irnos. Necesitamos insertar los manipuladores de eventos en las etiquetas `<A HREF>` de la imagen de navegación principal y crear después el mapa de imágenes que controle los submenús, además de colocarle también los manipuladores de eventos.

Inserción de los manipuladores de eventos

Vamos a ver las etiquetas `<A HREF>` de la navegación principal y colocar los manipuladores de eventos en su interior. Éste es el código de las imágenes de navegación principal y de la imagen del anuncio de texto/submenú.

```
<TABLE>
<TR>
<TD>
<A HREF="dirt/index.html"
onMouseOver="mainNavOn('dirt');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/dirt.gif" NAME="dirt" BORDER="0"
width="135" height="30" alt="Dirt"></A><BR>
<A HREF="money/index.html"
onMouseOver="mainNavOn('money');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/money.gif" NAME="money" BORDER="0"
width="135" height="27" alt="Money"></A><BR>
<A HREF="trends/index.html"
onMouseOver="mainNavOn('trends');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/trends.gif" NAME="trends" BORDER="0"
width="135" height="29" alt="Trends"></A><BR>
<A HREF="haute/index.html"
onMouseOver="mainNavOn('haute');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/haute.gif" NAME="haute" BORDER="0"
width="135" height="26" alt="Haute"></A><BR>
<A HREF="street/index.html"
onMouseOver="mainNavOn('street');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/street.gif" NAME="street" BORDER="0"
width="135" height="27" alt="Street"></A><BR>
<A HREF="profiles/index.html"
onMouseOver="mainNavOn('profiles');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/profiles.gif" NAME="profiles" BORDER="0"
width="135" height="29" alt="Profiles"></A><BR>
<A HREF="columns/index.html"
onMouseOver="mainNavOn('columns');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/columns.gif" NAME="columns" BORDER="0"
width="135" height="26" alt="Columns"></A><BR> </TD>
<TD>
<IMG SRC="images/nav/sec_blank.gif" NAME="submenu"
BORDER="0" USEMAP="#submenuMap" width="287" height="221"
alt="Submenu">
```

```
</TD>  
</TR></TABLE>
```

Los manipuladores de eventos `onMouseOver` deberían resultarle muy familiares, ya que se emplearon de la misma manera para otros *scripts* de *rollovers*. Los manipuladores `onMouseOut` se diferencian en que no pasamos el valor a la función. Esto se debe al modo diferente de manipular los *rollovers*. Observe también que aunque empleemos un mapa de imágenes para la imagen del anuncio de texto/submenú, necesitamos todavía darle un NAME adecuado en la etiqueta ``. Vamos a pasar a los últimos manipuladores de eventos: los que se encuentran dentro el mapa de imágenes. Este mapa de imágenes debe contener cuatro puntos calientes (uno por cada opción del submenú).

```
<MAP NAME="submenuMap">  
  
<AREA COORDS="22,10,278,48" SHAPE="rect"  
      HREF="JavaScript:goThere(3);"  
      onMouseOver="submenuOn(3);"  
      onMouseOut="mapOff();">  
  
<AREA COORDS="22,48,278,79" SHAPE="rect"  
      HREF="JavaScript:goThere(2);"  
      onMouseOver="submenuOn(2);"  
      onMouseOut="mapOff();">  
  
<AREA COORDS="22,79,278,116" SHAPE="rect"  
      HREF="JavaScript:goThere(1);"  
      onMouseOver="submenuOn(1);"  
      onMouseOut="mapOff();">  
  
<AREA COORDS="22,116,278,157" SHAPE="rect"  
      HREF="JavaScript:goThere(0);"  
      onMouseOver="submenuOn(0);"  
      onMouseOut="mapOff();">  
  
</MAP>
```

Necesitaremos tres manipuladores de eventos en cada uno de los puntos calientes. El manipulador `onClick` del campo `HREF` llamará a la función `goThere()`, que manipula-

ra el envío del usuario a la ubicación correcta cuando se seleccione una opción.

El manipulador `onMouseOver()` llama a la función `submenuOn()`, que se encargará de resaltar la imagen correcta cuando el usuario seleccione una opción. Los valores que pasamos a la función corresponden a las posiciones de las entradas en el segundo nivel del array multidimensional que hemos poblado anteriormente. Finalmente, el manipulador `onMouseOut` llama a la primera función `off`, es decir `mapOff()`.

Lo crea o no, ya hemos terminado. Los arrays pueden ser muy largos y embarazosos, pero hacen que la codificación sea más fácil y le permiten cambiar datos, como enlaces y nombres de sección, más fácilmente. Trataremos bastantes en este *script*, así que observe lo que hemos logrado y cómo lo hemos hecho.

ANÁLISIS DEL SCRIPT

1. Primero, creamos y poblamos las variables `whichImage` y `overImage`, que necesitaremos después en el *script* para vigilar lo que selecciona el usuario.
2. Despues, definimos y poblamos un array multidimensional para almacenar toda la información que necesitaremos para hacer que funcione el sistema de submenús.

```
<SCRIPT Language="JavaScript">  
<!-- El código a partir de aquí será ignorado por los<br/>→ navegadores más antiguos  
  
var whichMenu=" ";  
var overImage = 'no'  
section = new Array(4);  
section['dirt'] = new Array (4)  
section['dirt'][0] = 'supermodel'  
section['dirt'][1] = 'affairnet'
```

```

        section['dirt'][2] = 'moss'
        section['dirt'][3] = 'thrift'

section['money'] = new Array (4)
    section['money'][0] = 'worldclique'
    section['money'][1] = 'germany'
    section['money'][2] = 'mergers'
    section['money'][3] = 'nmearnings'

section['trends'] = new Array (4)
    section['trends'][0] = 'tshirts'
    section['trends'][1] = 'sweatshirts'
    section['trends'][2] = 'hats'
    section['trends'][3] = 'mugs'

section['haute'] = new Array (4)
    section['haute'][0] = 'paris'
    section['haute'][1] = 'gaudy'
    section['haute'][2] = 'taffeta'
    section['haute'][3] = 'royalty'

section['street'] = new Array (4)
    section['street'][0] = 'angry'
    section['street'][1] = 'prison'
    section['street'][2] = 'skater'
    section['street'][3] = 'hip_hop'

section['profiles'] = new Array (4)
    section['profiles'][0] = 'moxon'
    section['profiles'][1] = 'west'
    section['profiles'][2] = 'trinidadaddy'
    section['profiles'][3] = 'spurn'

section['columns'] = new Array (4)
    section['columns'][0] = 'column1'
    section['columns'][1] = 'column2'
    section['columns'][2] = 'column3'
    section['columns'][3] = 'column4'

```

3. Tras construir los arrays, creamos cinco funciones que necesitamos para ejecutar el *script*.

La primera función, `mainNavOn()`, se encarga de la desactivación de las imágenes de navegación principales seleccionadas, resalta la nueva imagen de navegación principal y coloca la imagen de submenú secundaria correspondiente.

```
function mainNavOn (sectionName)
```

```
{  
    overImage='yes';  
    // se asegura de que el usuario ya se está  
    // moviendo por encima de una imagen de nav. principal  
    if (whichMenu != " ")  
    {  
        // dejar de resaltar el nombre de sección antiguo  
        eval ("document." + whichMenu + ".src =  
            'images/nav/' + whichMenu + ".gif'");  
    }  
    // establecer variable al rollover nuevo  
    whichMenu = sectionName;  
    // resaltar la imagen nueva  
    eval( "document." + whichMenu + ".src = 'images/ nav/'  
        + whichMenu + "_over.gif'");  
    // hacer emergir imagen de navegación secundaria  
    eval("document_submenu.src = 'images/nav/' +  
        whichMenu + "_sec_blank.gif'");  
}
```

Después creamos la función submenuOn(), que determina el submenú que se despliega y coloca la categoría secundaria resaltada adecuada para ese submenú.

```
function submenuOn (sectionNum)  
{  
    overImage='yes';  
    if (whichMenu != " ")  
    {  
        // determina el nombre de subsección  
        // sobre la que se está moviendo el usuario  
        subSection = section[whichMenu][sectionNum];  
        // intercambia la imagen adecuada  
        eval("document_submenu.src = 'images/nav/' +  
            whichMenu + "_" + subSection + "_over.gif'");  
    }  
}
```

Las funciones tercera y cuarta comparten la responsabilidad de desactivar las diferentes imágenes de submenú cuando el usuario deja de moverse por encima de ellas. La primera de ellas, `mapOff()`, establece la variable `overImage` a `no` y después utiliza un temporizador para llamar a la segunda función, una vez haya transcurrido un segundo. La función `mapOff1()` se asegura de que el usuario no está en un gráfico y desactiva todos los gráficos resaltados.

```
function mapOff ()  
{  
    if (whichMenu != " ")  
    {  
        overImage = 'no';  
        setTimeout('mapOff2()',1000);  
    }  
}  
  
function mapOff2()  
{  
    if ((overImage == 'no') && (whichMenu != " "))  
    {  
        // recupera la imagen del artículo  
        document submenu.src = "images/nav/sec_blank.gif";  
  
        // cambiar la imagen de nav. principal  
        // por la versión no resaltada  
        eval ("document." + whichMenu + ".src =  
        'images/nav/" + whichMenu + ".gif'");  
  
        whichMenu = " ";  
    }  
}
```

La función `goThere()` envía al usuario al URL correcto cuando hace clic en uno de los puntos calientes de los mapas de imágenes. Verifica la categoría de navegación princi-

pal seleccionada y se introduce en el array para extraer el nombre de la sección secundaria que necesita para enviar al usuario al destino correcto.

```
function goThere (sectionNum)
{
    if (whichMenu != " ")
    {
        subSection = section[whichMenu][sectionNum]
        location.href = whichMenu + "/" + subSection +
        ".html"
    }
}
// Dejamos de ocultar el código aquí -->
</SCRIPT>
```

4. El paso final fue añadir manipuladores de eventos a las etiquetas <A HREF> de las imágenes de navegación principal, añadir el nombre adecuado de las imágenes afectadas por los *rollovers* e insertar los manipuladores necesarios dentro del mapa de imágenes que ejecuta los submenús.

```
<TABLE>
<TR>
<TD>
<A HREF="dirt/index.html"
onMouseOver="mainNavOn('dirt');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/dirt.gif" NAME="dirt" BORDER="0"
width="135" height="30" alt="Dirt"><BR>
<A HREF="money/index.html"
onMouseOver="mainNavOn('money');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/money.gif" NAME="money" BORDER="0"
width="135" height="27" alt="Money"><BR>
<A HREF="trends/index.html"
onMouseOver="mainNavOn('trends');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/trends.gif" NAME="trends" BORDER="0"
width="135" height="29" alt="Trends"><BR>
```

```
<A HREF="haute/index.html"
onMouseOver="mainNavOn('haute');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/haute.gif" NAME="haute" BORDER="0"
width="135" height="26" alt="Haute"></A><BR>

<A HREF="street/index.html"
onMouseOver="mainNavOn('street');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/street.gif" NAME="street" BORDER="0"
width="135" height="27" alt="Street"></A><BR>

<A HREF="profiles/index.html"
onMouseOver="mainNavOn('profiles');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/profiles.gif" NAME="profiles" BORDER="0"
width="135" height="29" alt="Profiles"></A><BR>

<A HREF="columns/index.html"
onMouseOver="mainNavOn('columns');"
onMouseOut="mapOff();">
<IMG SRC="images/nav/columns.gif" NAME="columns" BORDER="0"
width="135" height="26" alt="Columns"></A><BR> </TD>

<TD>

<IMG SRC="images/nav/sec_blank.gif" NAME="submenu"
BORDER="0" USEMAP="#submenuMap" width="287" height="221"
alt="Submenu">

</TD>

</TR></TABLE>

<MAP NAME="submenuMap">

<AREA COORDS="22,10,278,48" SHAPE="rect"
HREF="JavaScript:goThere(3);"
onMouseOver="submenuOn(3);"
onMouseOut="mapOff();">

<AREA COORDS="22,48,278,79" SHAPE="rect"
HREF="JavaScript:goThere(2);"
onMouseOver="submenuOn(2);"
onMouseOut="mapOff();">

<AREA COORDS="22,79,278,116" SHAPE="rect"
HREF="JavaScript:goThere(1);"
onMouseOver="submenuOn(1);"
onMouseOut="mapOff();">
```

```
<AREA COORDS="22,116,278,157" SHAPE="rect"
      HREF="JavaScript:goThere(0);"
      onMouseOver="submenuOn(0);"
      onMouseOut="mapOff();">
</MAP>
```

En este capítulo hemos aprendido algunos conceptos nuevos y hemos mejorado los conocimientos sobre conceptos que hemos tratado en capítulos anteriores. Veamos qué es lo que hemos tratado:

- Arrays multidimensionales; cómo definir, poblar y extraer datos de su interior.
- Un nuevo método de manipulación de *rollovers* de imagen.

Resumen

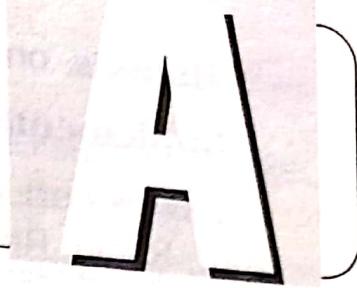
El *script* de este capítulo es un ejemplo de hasta dónde le puede llevar JavaScript. Combinamos muchos elementos de capítulos anteriores para crear una página que está en la cima de la programación web. Pero no se crea que ésta es la aplicación más complicada de JavaScript. DHTML y las hojas de estilo comienzan a ganar popularidad, las nuevas funciones que se han añadido a JavaScript son verdaderamente alucinantes. Hemos creado algunos *scripts* fabulosos, aunque fuesen para dos compañías imaginarias. Ahora es el momento de ver lo que ha aprendido y emplear los conocimientos adquiridos para trabajar con sitios propios.

Proyectos avanzados

1. Cree un *script* que rote aleatoriamente la imagen de una bandera y haga que cada gráfico vaya a un URL diferente.

2. Cree un *script* que realice un envío seguro de un formulario encriptando los datos de los campos del mismo.
3. Cree un *script* que guarde los datos de la información del usuario en un archivo del servidor web, para que pueda servirle una página personalizada cuando regrese.
4. Para los sitios web que tienen muchos gráficos en la página principal, cree una pantalla de bienvenida y escriba un *script* que cargue todos los gráficos necesarios para que la página principal se abra mucho más rápido.

Manipuladores de eventos



JavaScript es un lenguaje que manipula eventos; es decir, los *scripts* creados en JavaScript se ejecutan normalmente como resultado de un evento que puede ser originado tanto por el usuario como por el navegador. Podemos aparejar estos eventos mediante la utilización de los manipuladores de eventos. Este apéndice se creó con la intención de ofrecerle un lugar donde comprobar los diferentes manipuladores de eventos, su sintaxis, así como una descripción de sus diversos usos.

onAbort

Descripción: este manipulador se lanza cuando el usuario aborta la carga de una imagen, haciendo clic, o bien pulsando el botón Detener.

Sintaxis: onAbort="código que se ejecutará"

Implementado en: Navigator 3.0

Utilizado con: Imágenes.

onBlur

Descripción: se lanza cuando un marco, una ventana o un elemento de formulario pierde el foco.

Sintaxis: onBlur="código que se ejecutará"

Implementado en: Navigator 2.0/3.0

Usado con: Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window

onChange

Descripción: se lanza cuando un campo Select, Text, o Textarea pierde el foco y cambia su valor.

Sintaxis: onChange="código que se ejecutará"

Implementado en: Navigator 2.0/3.0

Usado con: FileUpload, Select, Text, Textarea

onClick

Descripción: se lanza al hacer clic en un objeto Form, o cuando el manipulador se coloca dentro de una etiqueta <HREF> y se hace clic en la imagen o el enlace del texto.

Sintaxis: onClick="código que se ejecutará"

Implementado en: Navigator 2.0/3.0

Usado con: Button, Checkbox, document, Link, Radio, Reset, Submit

onDbClick

Descripción: se lanza al hacer doble clic en un objeto Form, o cuando el manipulador se coloca dentro de una etiqueta <HREF> y se hace doble clic en el enlace.

Sintaxis: onDbClick="código que se ejecutará"

Implementado en: Navigator 4.0

Usado con: document, Link

onDragDrop

Descripción: se lanza cuando el usuario suelta un objeto, como un archivo, dentro de la ventana de Navigator.

Sintaxis: onDragDrop = "código que se ejecutará"

Implementado en: Navigator 4.0

Usado con: Window

onError

Descripción: se lanza cuando la carga de un documento o imagen origina un error.

Sintaxis: onError = "código que se ejecutará"

Implementado en: Navigator 3.0

Usado con: Image, Window

onFocus

Descripción: se lanza cuando un elemento Window, Frame, Frameset o Form recibe el foco.

Sintaxis: onFocus = "código que se ejecutará"

Implementado en: Navigator 2.0/3.0/4.0

Usado con: Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window

onKeyDown

Descripción: se lanza cuando el usuario pulsa una tecla.

Sintaxis: onKeyDown = "código que se ejecutará"

Implementado en: Navigator 4.0

Usado con: document, Image, Link, Textarea

onKeyPress

Descripción: se lanza cuando el usuario pulsa una tecla.

Sintaxis: onKeyPress = “código que se ejecutará”

Implementado en: Navigator 4.0

Usado con: document, Image, Link, Textarea

onKeyUp

Descripción: se lanza cuando el usuario suelta una tecla.

Sintaxis: onKeyUp = “código que se ejecutará”

Implementado en: Navigator 4.0

Usado con: document, Image, Link, Textarea

onLoad

Descripción: se lanza cuando termina de cargarse todo el contenido de una ventana o todos los marcos dentro de un conjunto de marcos. También se puede lanzar cargando una imagen.

Sintaxis: onLoad = “código que se ejecutará”

Implementado en: Navigator 2.0/3.0

Usado con: Image, Layer, Window

onMouseDown

Descripción: se lanza cuando el usuario pulsa el botón del ratón.

Sintaxis: onMouseDown = “código que se ejecutará”

Implementado en: Navigator 4.0

Usado con: Button, document, Link

onMouseMove

Descripción: se lanza cuando el usuario mueve el cursor.

Sintaxis: onMouseMove = "código que se ejecutará"

Implementado en: Navigator 4.0

Usado con: ninguno.

onMouseOut

Descripción: se lanza cuando el usuario mueve el cursor fuera del objeto.

Sintaxis: onMouseOut = "código que se ejecutará"

Implementado en: Navigator 3.0

Usado con: Layer, Link

onMouseOver

Descripción: se lanza cuando el usuario mueve el cursor por encima del objeto.

Sintaxis: onMouseOver = "código que se ejecutará"

Implementado en: Navigator 2.0/3.0

Usado con: Layer, Link

onMouseUp

Descripción: se lanza cuando el usuario suelta el botón del ratón.

Sintaxis: onMouseUp = "código que se ejecutará"

Implementado en: Navigator 4.0

Usado con: Button, document, Link

onMove

Descripción: se lanza cuando un *script* o el usuario mueve la ventana o un marco.

Sintaxis: `onMove = "código que se ejecutará"`

Implementado en: Navigator 4.0

Usado con: Window

onReset

Descripción: se lanza cuando el usuario restaura un formulario.

Sintaxis: `onReset = "código que se ejecutará"`

Implementado en: Navigator 3.0

Usado con: Form

onSelect

Descripción: se lanza cuando el usuario selecciona un texto desde un campo de formulario Text o Textarea.

Sintaxis: `onSelect = "código que se ejecutará"`

Implementado en: Navigator 2.0

Usado con: Text, Textarea

onSubmit

Descripción: se lanza cuando el usuario envía un formulario.

Sintaxis: `onSubmit = "código que se ejecutará"`

Implementado en: Navigator 2.0

Usado con: Form

onUnload

Descripción: se lanza cuando el usuario abandona un documento.

Sintaxis: onUnload = "código que se ejecutará"

Implementado en: Navigator 2.0

Usado con: Window

Objetos de JavaScript

B

Como JavaScript es un lenguaje orientado a objetos y es un poco complicado sin conocer los objetos que soporta, nos hemos decidido a incluir esta referencia. Este apéndice divide los distintos objetos de JavaScript en cinco categorías, e indica cuáles son sus propiedades y métodos.

Objetos esenciales

No están asociados con la jerarquía de objetos de JavaScript y están disponibles en las aplicaciones del lado del cliente y del lado del servidor.

Array

Descripción: un array es un objeto que permite al usuario almacenar conjuntos de datos, con cada elemento del conjunto de datos en una posición única, que a su vez puede ser referenciado o recuperado.

Sintaxis de creación: new Array(arrayLong); o new Array(elemento0, elemento1, ..., elementoN);

Parámetros: arrayLong. Longitud inicial deseada para el array. ElementN: Conjunto inicial de valores que será alma-

cenado en el array. La longitud del array se establecerá de acuerdo al número de argumentos.

Implementado en: Navigator 3.0

Propiedades: index, input, length, prototype

Métodos: concat, join, pop, push, reverse, shift, slice, splice, toString, unshift

Boolean

Descripción: el objeto Boolean se emplea como un contenedor para un valor *booleano*.

Sintaxis de creación: new Boolean(valor).

Parámetros: valor. Valor inicial del objeto *booleano*.

Implementado en: Navigator 3.0

Propiedades: prototype

Métodos: toString

Date

Descripción: otorga al usuario la capacidad de trabajar con fechas y horas.

Sintaxis de creación: new Date(); o new Date("mes día, año horas:minutos:segundos"); o new Date(año_núm, mes_núm, día_núm, hora_núm, min_núm, seg_núm);

Parámetros: día, horas, minutos, mes, segundos, año. Si se emplean, estas partes de la fecha serán valores de cadena.

día_núm, hora_núm, min_núm, mes_núm, seg_núm y año_núm. Si se usan, estas partes de la fecha serán valores enteros.

Implementado en: Navigator 2.0/3.0

Propiedades: prototype

Métodos: getDate, getDay, getHours, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getYear, parse, setDate, setHours, setMinutes, setMonth, setSeconds, setTime, setYear, toGMTString, toLocaleString, UTC

Function

Descripción: este objeto contiene líneas de JavaScript que se ejecutan cuando se accede al objeto.

Sintaxis de creación: function name(arg1, arg2, ...argN)
{ cuerpo de la función }

Parámetros: arg1, arg2, ...argN. Puede utilizar un conjunto de valores de cadena para almacenar datos que se pasen a la función.

cuerpo de la función. Conjunto de comandos JavaScript que interpreta la función.

Implementado en: Navigator 3.0

Propiedades: arguments, arity, caller, prototype

Métodos: toString

Math

Descripción: este objeto contiene métodos y propiedades que ayudan en las matemáticas avanzadas.

Sintaxis de creación: Ninguna. Este objeto está integrado en el motor de JavaScript y puede llamarlo o hacer referencia a él sin tener que crearlo.

Implementado en: Navigator 2.0

Propiedades: E, LN10, LN2, LOG10E, LOG2E, PI, SQRT1, SQRT2

Métodos: abs, acos, asin, atan, atan2, ceil, cos, exp, floor, log, max, min, pow, random, round, sin, sqrt, tan

Number

Descripción: contiene valores numéricos primitivos. Es muy útil cuando se trabaja con valores numéricos.

Sintaxis de creación: new Number(valor);

Parámetros: valor. Valor numérico que contiene el objeto.

Implementado en: Navigator 3.0/4.0

Propiedades: MAX VALUE, MIN VALUE, NaN, NEGATIVE INFINITY, POSITIVE INFINITY, prototype

Métodos: toString

Object

Descripción: está integrado en JavaScript y es el objeto del cual descienden todos los demás.

Sintaxis de creación: new object();

Parámetros: ninguno.

Implementado en: Navigator 2.0/3.0

Propiedades: constructor, prototype

Métodos: eval, toString, unwatch, valueOf, watch

RegExp

Descripción: contiene una expresión regular que se puede utilizar para encontrar, reemplazar y manipular coincidencias y cadenas.

Sintaxis de creación: new RegExp("patrón", "indicadores");

Parámetros: patrón. El texto que contiene la expresión regular.

indicadores: hay tres valores posibles para un indicador: global match(g), ignore case(i), y ambos, global match e ignore case(gi).

Implementado en: Navigator 4.0

Propiedades: \$n, \$, \$*, \$&, \$+, &', \$', global, ignoreCase, input, lastIndex, lastMatch, lastParen, leftContext, multiline, rightContext, source

Métodos: compile, exec, test

String

Descripción: contiene una serie de caracteres que crean una cadena.

Sintaxis de creación: new String(cadena);

Parámetros: cadena. Una cadena.

Implementado en: Navigator 2.0/3.0/4.0

Propiedades: length, prototype

Métodos: anchor, big, blink, bold, charAt, charCodeAt, concat, fixed, fontcolor, fontsize, fromCharCode, indexOf, italics, lastIndexOf, link, match, replace, search, slice, small, split, strike, sub, substr, substr, sup, toLowerCase, toUpperCase

Objetos document

Esta sección está dedicada al objeto document y sus objetos relacionados.

Anchor

Descripción: este objeto contiene una cadena que es el destino de un enlace de hipertexto contenido en una página HTML.

Sintaxis de creación: cadena.anchor(nombreAtributo)

Parámetros: cadena. Un objeto de cadena.

nombreAtributo. Una cadena que especifica un nombre.

Creado por: se creará un objeto Anchor bien mediante la etiqueta <A> en un documento HTML, o bien llamando al método String.anchor.

Implementado en: Navigator 1.0

Manipuladores de eventos: NA

Propiedades: ninguna.

Métodos: watch, unwatch

Applet

Descripción: contiene cualesquiera *applets* Java incluidos en una página HTML.

Sintaxis de creación: NA

Creado por: la etiqueta applet de HTML.

Implementado en: Navigator 3.0

Propiedades: todas las propiedades públicas del *applet* están disponibles mediante el objeto.

Métodos: todos los métodos públicos.

Area

Descripción: representa un área de un mapa de imágenes. Si desea más información sobre sus propiedades, consulte el objeto Link.

Implementado en: Navigator 3.0

document

Descripción: contiene las propiedades del documento actual.

Sintaxis de creación: NA

Creado por: se crea mediante la etiqueta <BODY> de un documento HTML mientras el motor en tiempo de ejecución lee la página.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseUp

Propiedades: alinkColor, anchors, applets, bgColor, cookie, domain, embeds, fgColor, formName, forms, images, lastModified, layers, linkColor, links, plugins, referrer, title, URL, vlinkColor

Métodos: captureEvents, close, getSelection, handleEvent, open, releaseEvents, routeEvent, write, writeln

Image

Descripción: contiene una imagen, y permite el acceso a las propiedades de la misma.

Sintaxis de creación: new Image(anchura, altura)

Parámetros: anchura. La anchura de la imagen.

altura. La altura de la imagen.

Creado por: el constructor de la imagen o una etiqueta que se encuentra en el documento HTML.

Implementado en: Navigator 3.0/4.0

Manipuladores de eventos: onAbort, onError, onKeyDown, onKeyPress, onKeyUp, onLoad

Propiedades: border, complete, height, hspace, lowsrc, name, prototype, src, vspace, width

Métodos: handleEvent

Layer

Descripción: contiene una capa de un documento HTML y permite el acceso a las propiedades de las capas.

Sintaxis de creación: NA

Creado por: Las etiquetas <LAYER> o <ILAYER> de un documento HTML pueden crear un objeto Layer.

Implementado en: Navigator 4.0

Manipuladores de eventos: onBlur, onFocus, onLoad, onMouseOut, onMouseOver

Propiedades: above, background, below, bgColor, clip.bottom, clip.height, clip.left, clip.right, clip.top, clip.width, document, left, name, pageX, pageY, parentLayer, siblingAbove, siblingBelow, src, top, visibility, zIndex

Métodos: captureEvents, handleEvent, load, moveAbove, moveBelow, moveBy, moveTo, moveToAbsolute, releaseEvents, resizeBy, resizeTo, routeEvent

Link

Descripción: contiene un enlace a un documento HTML y permite el acceso a las propiedades del enlace.

Sintaxis de creación: cadena.link(href)

Parámetros: cadena. Un objeto de cadena.

 HREF. Una cadena que especifica un URL.

Creado por: las etiquetas <A HREF> o <AREA> de un documento HTML o una llamada al método String.link crean un objeto Link.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseOut, onMouseOver, onMouseUp

Propiedades: hash, host, hostname, href, pathname, port, protocol, search, target, text

Métodos: handleEvent

Objetos Window

Esta sección se refiere al objeto Window y todos los objetos relacionados con el mismo.

Frame

Descripción: este objeto contiene un marco, especificado en un conjunto de marcos HTML. Cada marco es, en realidad, un objeto Window; el objeto Frame sólo se usa por comodidad.

Sintaxis para su creación: NA

Creado por: tanto la etiqueta <FRAME> como la etiqueta <FRAMESET> en un documento HTML originarán un objeto Frame.

Implementado en: Navigator 2.0/3.0

Manipuladores de eventos: véase el objeto Window.

Propiedades: véase el objeto Window.

Métodos: véase el objeto Window.

History

Descripción: este objeto contiene un array que almacena todos los URL que el usuario ha visitado dentro de esa ventana.

Sintaxis para su creación: NA

Creado por: éste es un objeto integrado en JavaScript.

Implementado en: Navigator 2.0/3.0

Manipuladores de eventos: NA

Propiedades: current, length, next, previous

Métodos: back, forward, go

Location

Descripción: este objeto contiene el URL actual.

Sintaxis para su creación: NA

Creado por: éste es un objeto integrado en JavaScript.

Implementado en: Navigator 2.0/3.0

Manipuladores de eventos: NA

Propiedades: hash, host, hostname, href, pathname, port, protocol, search

Métodos: reload, replace

screen

Descripción: este objeto contiene información de la pantalla de visualización y los colores.

Sintaxis para su creación: NA

Creado por: es un objeto integrado en JavaScript que crea el motor en tiempo de ejecución.

Implementado en: Navigator 4.0

Manipuladores de eventos: NA

Propiedades: availHeight, availWidth, height, pixelDepth, width

Métodos: NA

Window

Descripción: este objeto describe un navegador o un marco.

Sintaxis para su creación: NA

Creado por: este objeto es creado por cada etiqueta HTML <BODY>, <FRAMESET> y <FRAME>, o bien por el método open del objeto Window.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onDragDrop, onError, onFocus, onLoad, onMove, onResize, onUnload

Propiedades: closed, defaultStatus, document, frames, history, innerHeight, innerWidth, length, location, menubar, name, opener, outerHeight, outerWidth, pageXOffset, pageYOffset, parent, personalbar, scrollbars, self, status, statusbar, toolbar, top, window

Métodos: alert, back, blur, captureEvents, clearInterval, clearTimeout, close, confirm, disableExternalCapture, enableExternalCapture, find, focus, forward, handleEvent, home, moveBy, moveTo, open, print, prompt, releaseEvents, res-

zeBy, resizeTo, routeEvent, scroll, scrollBy, scrollTo, setInterval, setTimeout, stop

Objetos Form

Esta sección está dedicada al objeto Form y todos sus objetos relacionados.

Button

Descripción: este objeto contiene un botón de un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <INPUT> con “button” especificado como atributo TYPE.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onClick, onFocus, onMouseDown, onMouseUp

Propiedades: form, name, type, value

Métodos: blur, click, focus, handleEvent

Checkbox

Descripción: este objeto contiene una casilla de verificación del formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <INPUT> con “checkbox” especificado como atributo TYPE.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onClick, onFocus

Propiedades: checked, defaultChecked, form, name, type, value

Métodos: blur, click, focus, handleEvent

FileUpload

Descripción: este objeto contiene un elemento upload de archivo de un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <INPUT> con "file" especificado como atributo TYPE.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onChange, onFocus

Propiedades: form, name, type, value

Métodos: blur, focus, handleEvent, select

Form

Descripción: este objeto contiene un formulario HTML y todos los objetos contenidos en ese formulario.

Sintaxis para su creación: NA

Creado por: el motor en tiempo de ejecución crea este objeto cuando aparece en una etiqueta <FORM> en un documento HTML.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onReset, onSubmit

Propiedades: action, elements, encoding, length, method, name, target

Métodos: handleEvent, reset, submit

Hidden

Descripción: este objeto contiene un objeto `Text` oculto de un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta `<INPUT>` con “hidden” especificado como atributo `TYPE`.

Implementado en: Navigator 2.0/3.0

Manipuladores de eventos: NA

Propiedades: `form`, `name`, `type`, `value`

Métodos: NA

Option

Descripción: este objeto contiene una opción individual de un menú desplegable de selección.

Sintaxis para su creación: `new Option(texto, valor, opciónPredeterminada, seleccionado)`

Parámetros: `texto`. Establece el texto que aparece en la lista del menú.

`valor`. Establece los valores que vuelven al servidor cuando el formulario es enviado con la opción elegida.

`opciónPredeterminada`. Establece la opción seleccionada inicialmente.

`seleccionado`. Establece el estado actual de la opción.

Creado por: la etiqueta `<OPTION>` o mediante el uso del constructor de `Option`.

Implementado en: Navigator 2.0/3.0

Manipuladores de eventos: NA

Propiedades: `defaultSelected`, `selected`, `text`, `value`

Métodos: NA

password

Descripción: este objeto contiene un campo de texto de un formulario HTML que oculta su valor.

Sintaxis para su creación: NA

Creado por: la etiqueta <INPUT> con “password” especificado como atributo TYPE.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onFocus

Propiedades: defaultValue, form, name, type, value

Métodos: blur, focus, handleEvent, select

radio

Descripción: este objeto contiene un único botón de radio de un conjunto de botones en un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <INPUT> con “radio” especificado como atributo TYPE.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onClick, onFocus

Propiedades: checked, defaultChecked, form, name, type, value

Métodos: blur, click, focus, handleEvent

reset

Descripción: este objeto contiene un botón Restaurar de un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <INPUT> con “reset” especificado como atributo TYPE.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onClick, onFocus

Propiedades: form, name, type, value

Métodos: blur, click, focus, handleEvent

Select

Descripción: este objeto contiene un menú desplegable de selección de un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <SELECT> insertada en un formulario HTML.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onChange, onFocus

Propiedades: form, length, name, options, selectedIndex, type

Métodos: blur, focus, handleEvent

Submit

Descripción: este objeto contiene un botón Enviar de un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <INPUT> con “submit” especificado como atributo TYPE.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onClick, onFocus

Propiedades: form, name, type, value

Métodos: blur, click, focus, handleEvent

Text

Descripción: este objeto contiene un campo texto de un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <INPUT> con "text" especificado como atributo TYPE.

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onChange, onFocus, onSelect

Propiedades: defaultValue, form, name, type, value

Métodos: blur, focus, handleEvent, select

Textarea

Descripción: este objeto contiene un campo texto de entrada múltiple de un formulario HTML.

Sintaxis para su creación: NA

Creado por: la etiqueta <TEXTAREA>

Implementado en: Navigator 2.0/3.0/4.0

Manipuladores de eventos: onBlur, onChange, onFocus, onKeyDown, onKeyPress, onKeyUp, onSelect

Propiedades: defualtValue, form, name, type, value

Métodos: blur, focus, handleEvent, select

Objetos Browser

Esta sección está dedicada a los objetos que contienen las propiedades específicas del navegador.

MimeType

Descripción: este objeto contiene un tipo MIME soportado por el navegador.

Sintaxis para su creación: NA

Creado por: este objeto se crea automáticamente cuando se carga el navegador.

Implementado en: Navigator 3.0

Propiedades: description, enabledPlugin, suffixes, type

Métodos: NA

Navigator

Descripción: este objeto contiene información acerca del navegador del usuario.

Sintaxis para su creación: NA

Creado por: este objeto se crea automáticamente cuando se carga el navegador.

Implementado en: Navigator 2.0/3.0/4.0

Propiedades: appCodeName, appName, appVersion, language, mimeTypes, platform, plugins, userAgent

Métodos: javaEnabled, plugins.refresh, preference, taintEnabled

Plugin

Descripción: este objeto contiene un *plug-in* instalado en el navegador del usuario.

Sintaxis para su creación: NA

Creado por: se crea automáticamente cuando se carga el navegador.

Implementado en: Navigator 3.0

Propiedades: description, filename, length, name

Métodos: NA

Índice alfabético

&&, 131

@, 126

\r, 124, 149

||, 123, 131

=, operador de asignación, 8, 39

A

Adelante, botón, 168

arrays

creación, 80-85, 153-156,
189-193

función de población, 85-99

longitud, 104

población, 82-85, 189-193

Atrás, botón, 168

atributos

alt, 37

Height, 37

LANGUAGE, 4, 8

NAME, 32-33, 51, 62-63, 208

SIZE, 14

SRC, 25

VALUE, 73

Width, 37

B

bucles

while, 87-99, 124, 133-136, 138,
146, 172

C

campos vacíos en formularios,

comprobar, 114-125

códigos postales, comprobar,
131-139

conjunto de marcos (FRAMESET),
169-171, 176

contraseña, 109-111

correo electrónico, comprobar
direcciones, 126-128

creación de objetos IMAGE, 29

E

etiquetas

A, 46-47, 49-50, 62, 63, 158,
206, 213

ANCHOR, 42

BODY, 161, 175

FONT, 2, 13-14

FORM, 71, 121

HEAD, 7, 161

HREF, 162

IMG, 32-33, 37, 49-51, 62, 208

SCRIPT, 3-4, 8, 18, 25, 29, 71,

F

fecha de última modificación, 16-24
 formularios
 comprobar campos vacíos, 114-125
 comprobar códigos postales con números, 131-139
 comprobar datos erróneos, 126-139
 comprobar teléfono, 128-131
 informar de lo que es erróneo, 140-149
 FRAMESET, 169-171, 176
 funciones, 38-41
 alert(), 141
 comprobar información de un formulario, 120-124
 Checker(), 177-178
 errorCheck(), 145
 goThere(), 213
 Highlighter(), 174, 179
 limpiar un menú, 91-96
 mainNavOn(), 210
 mapOff(), 209, 212
 navegación, 99-100
 off(), 173, 202, 205
 PageCreator(), 156-161, 162
 población de un array, 85-99
 SectionChecker(), 174-178
 submenuOn(), 198, 211
 submenús, 193-206

G

gif animados, 67
 Go, botón, 73-76
 gráficos, desactivar, 56

H

HTML

código para menús desplegables, 90-91
 dinámico, 13-14
 enviar a otra página, 77
 formulario, 116-119
 FRAMESET, 169-171, 176
 insertar un *script*, 3-4

I

imprimir dinámicamente datos que no son cadenas de caracteres, 20-22
 instrucciones, véase sentencias
 Ir, véase Go, botón

J

JavaScript
 comprobar información del visitante, 119-124
 comprobar prefijo en teléfono, 128-131
 comunicación entre marcos, 168-181
 función de navegación, 71-72
 informar de lo que es erróneo, 140-149
 manipuladores de eventos, 206-209
 mensajes de error, 140-143
 menús desplegables, 70-79
 página de inicio de sesión, 108-111
 rollovers, 27-68
 varios menús para navegar, 79-108
 jerarquías, 5-6

L

Login, 110

M

manipuladores de eventos, 41-43, 175-178
inserción, 49-51
onAbort, 218
onBlur, 218-219
onClick, 73, 77, 78-79, 119, 162, 218
onChange, 74, 76, 78-79, 110, 218
onDblClick, 218
onDragDrop, 219
onError, 219
onFocus, 219
onKeyDown, 219
onKeyPress, 219
onKeyUp, 219
onLoad, 175, 177-178, 220
onMouseDown, 220
onMouseMove, 221
onMouseOut, 41-43, 62, 208, 221
onMouseOver, 41-42, 62, 208, 221
onMouseUp, 221
onMove, 222
onReset, 222
onSelect, 222
onSubmit, 222
onUnload, 223
marcos, 168-181
menús desplegables, 70-79
comprobar que se selecciona una opción, 124-125
entrada predeterminada, 91
función de navegación, 71-72
manipuladores, 73-74, 100-102
varios menús para navegar, 79-108
métodos
close(), 162

indexOf(), 127
open(), 159
setTimeout(), 60, 202
substr(), 10, 15, 137
write(), 166

O

objetos
Array, 225-226
Boolean, 226
Browser, 241-243
MimeType, 242
Navigator, 242
Plugin, 242-243
Date, 226-227
DOCUMENT, 5, 17-18, 72, 78, 152, 166, 168, 206, 221-225
Anchor, 230
Applet, 230
Area, 230-231
Document, 231
Image, 231-232
Layer, 232
Link, 232-233
Form, 235-241
Button, 236
Checkbox, 236
FileUpload, 236-237
Form, 237
Hidden, 237-238
Option, 238
Password, 238-239
Radio, 239
Reset, 239-240
Select, 240
Submit, 240
Text, 241
Textarea, 241
FRAME, 168, 170
Function, 227
IMAGE, 28-67, 127, 172, 192, 195

creación de objetos, 52-54
 Math, 227-228
 NAVIGATOR, 6, 10, 25
 navigator.platform, 15
 Number, 228
 Object, 228
 RegExp, 228-229
 STRING, 127, 228
 WINDOW, 5, 42, 152, 158, 165,
 167, 168, 233-235
 Frame, 233
 History, 234
 Location, 234
 screen, 234-235
 Window, 235
 operadores
 asignación (=), 9, 39
 OR, disyuntiva, 123

P

parent, 171
 pausas, 60
 Perl, 109
 plataformas, diferencias entre, 1-2
 código específico, 2-3
 DETECCIÓN, 6-13
 prefijo, comprobación de su
 existencia, 128-131
 propiedades
 lastModified, 17-20
 platform, 6-10
 source, 6, 35, 39, 64, 195
 src, 39-41, 170
 SUBMIT, 111
 text, 97, 98
 value, 97, 98

R

retorno de carro (\r), 124, 149
rollovers, 27-68
 alt, atributo, 37

creación de objetos IMAGE,
 52-54
 funcionalidad avanzada, 47-66
 funciones, 37-41, 54-61
 Height, atributo, 37
 manipuladores de eventos,
 41-51
 objetos On y Off, 34-36
 peculiaridades, 66-67
 Width, atributo, 37

S

sentencias

document.write(), 13-14, 159,
 161, 162
 document.writeln(), 13-14, 15,
 20, 21, 25
 eval(), 40, 98, 157
 function, 38
 if, 7, 10-13, 13, 15, 20, 30-31,
 39, 86, 89-91
 y la disyuntiva OR, 123
 return, 43
 var, 9-10
 while, 87-99, 124, 136, 138, 146,
 172
 submenús, 193-206
 Submit, botón, 119

T

teléfono, comprobación del
 prefijo, 128-131
 top, 171

V

variables, 9-10
 ventanas, 152-167
 abrir, 158-159
 comunicación entre marcos,
 168-181