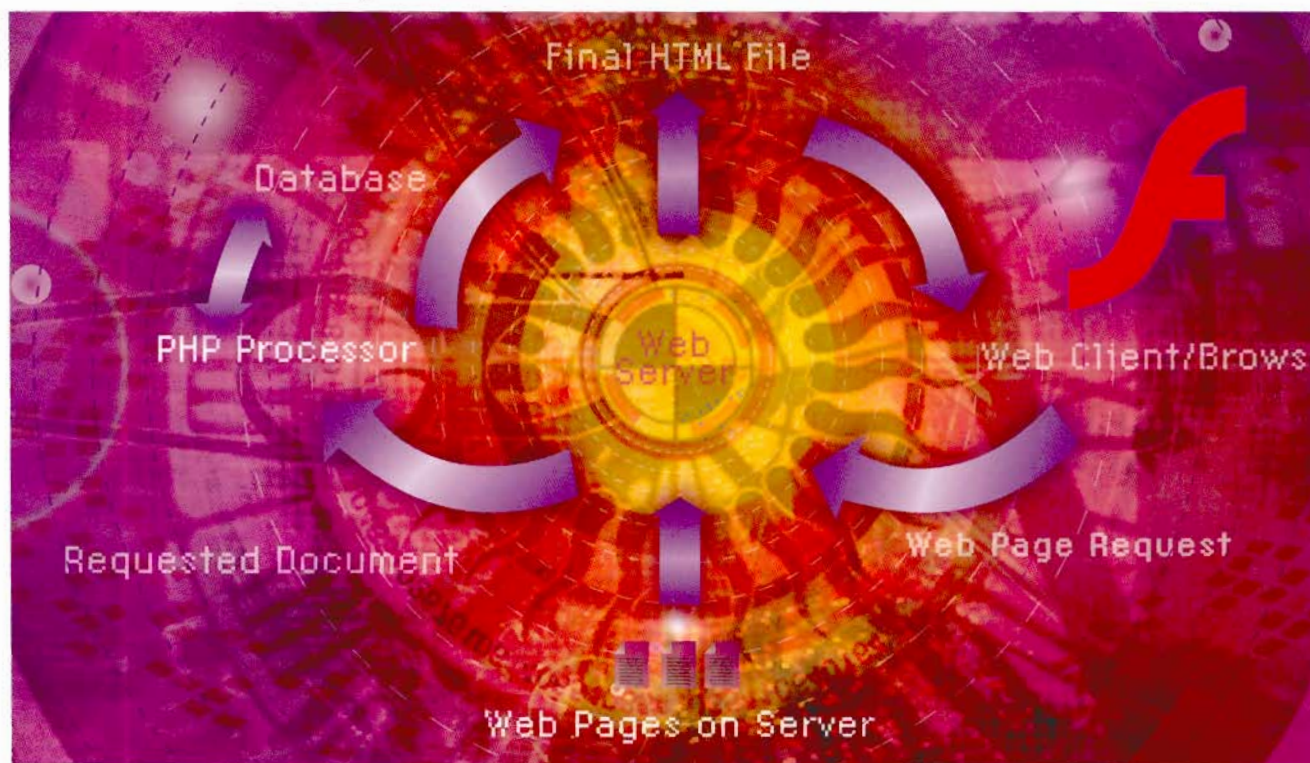


Flash, PHP y MySQL

Contenidos dinámicos

Integre con éxito las tres aplicaciones.



DANIEL DE LA CRUZ HERAS
CARLOS ZUMBADO RODRÍGUEZ

ANAYA
MULTIMEDIA
DISEÑO Y CREATIVIDAD



Incluye
CD-ROM

Índice de contenidos

Introducción	15
Capítulo 1. Flash y contenidos dinámicos	21
Nociones básicas de ActionScript para Flash MX	23
Trabajar en Modo Experto de edición	23
Variables	26
Arrays	28
La ventana de Salida	29
Funciones	31
delete	33
Jerarquías de clips	34
_global	35
Condiciones	36
Bucles	38
Objetos	40
Clips de película creados dinámicamente. Dibujar mediante código	41
Campos de texto creados dinámicamente. Formatos de texto	43
Carga dinámica de documentos Flash (.swf)	45
Carga dinámica de imágenes (.jpg)	46
Carga dinámica de sonidos (.mp3)	47

Carga dinámica de datos almacenados en ficheros de texto (.txt)	51
Carga dinámica de datos almacenados en ficheros escritos en lenguajes de marcado XML (.xml)	54

Capítulo 2. Instalación de servidores 69

Instalación de servidores en ordenadores con Windows	69
Instalación de servidores en ordenadores Macintosh	75
Activación del módulo de PHP	76
Iniciar y detener el servidor Web Apache	78
Creación de una cuenta MySQL	81
Instalación de la base de datos MySQL	83
Instalación y puesta en marcha de phpMyAdmin	88

Capítulo 3. PHP 93

¿Qué es PHP?	93
Características del lenguaje	94
Características generales de PHP	94
Impresión de datos	94
Utilización de variables en PHP	95
Variables predefinidas	95
Variables locales, globales y superglobales	95
Concatenación de cadenas de texto	96
Adición de más texto a una variable ya existente	97
Utilización y manipulación de arrays	97
Transformaciones de cadenas de texto en arrays y viceversa	102
Función para manejar fechas (date)	103
Imprimir la fecha en castellano	104
Impresión de fechas pasadas o futuras	106
Manipulación de ficheros	106
Abrir un fichero	106
Lectura de un fichero	107
Escribir en un fichero	108
Cerrar un fichero	108
Ejemplo práctico: contador basado en fichero de texto	108
Envío de correos utilizando PHP	109
Más información sobre PHP	110

Capítulo 4. MySQL..... 113

Servidores de bases de datos	113
Organización de MySQL	113
Columnas	113

Datos	119
Índices	119
Manipulación y utilización de MySQL	119
Creación de una base de datos	120
Creación de una tabla para la base de datos	121
Inserción de información	124
Selección de registros de la tabla	126
Modificación de registros almacenados en la tabla	130
Eliminación de registros almacenados en la tabla	132
Más información sobre MySQL	132

Capítulo 5. MySQL y PHP 135

MySQL y PHP	135
Ficheros de configuración y acciones repetitivas	136
Obtener siempre una respuesta	136
Uso de MySQL con PHP (identificadores y arrays)	137
Establecer una conexión con MySQL	137
Selección de la base de datos	138
Creación de los ficheros de configuración	139
Realización de consultas a MySQL	140
Impresión de resultados	140
Liberación de memoria	147
Cierre de la conexión con MySQL	148
Modificación de la información de la base de datos	148
Envío de variables usando POST y GET	149
Listado de registros	152
Obtención de detalles de un registro	157
Modificación de un registro	162
Inserción de un nuevo registro	170
Borrado de un registro específico	174

Capítulo 6. Flash, PHP y MySQL 181

Primeros pasos	182
Cómo utilizar el material de este capítulo	182
Obtención de datos usando el objeto LoadVars	183
Ejemplo básico	184
Obtención del listado de usuarios	186
Utilización del objeto LoadVars y dos documentos PHP	190
Utilización del objeto LoadVars y un solo documento PHP	199
Obtención de datos utilizando el objeto XML	206
Ejemplo básico	206
Obtención del listado de usuarios	207

Utilización del objeto XML y dos documentos PHP	210
Utilización del objeto XML y un documento PHP	217
Obtención de datos desde un documento XML	222

Capítulo 7. Un caso práctico 237

Creación de aplicación para subir ficheros al servidor	238
Creación de una galería de imágenes	242
Creación de zonas de acceso restringido	245
Agenda: Administrador de contenidos + Interfaz	254
Estructura del proyecto	254
Tabla agenda	255
Carpeta includes	256
Carpeta admin	262
Realización en Flash de la interfaz para la agenda	278

Capítulo 8. Recursos y enlaces 301

Apéndice Contenido del CD-ROM 307

Índice alfabético 311

Introducción

Los contenidos que observamos diariamente en Internet han sufrido una profunda transformación en los últimos años, ocasionada por el propio desarrollo de la tecnología que los soporta. Si hace un tiempo los sitios Web constaban exclusivamente de distintas páginas escritas en formato HTML en las que se incluía texto e imágenes, hoy día es corriente que el sitio Web en sí no sea más que la parte visible de una aplicación, instalada en el servidor, cuyo funcionamiento consiste en extraer información de una base de datos y dotarla de una apariencia apta para ser mostrada en el navegador de Internet del usuario que está visitando ese sitio Web, gracias a la combinación entre el propio lenguaje HTML y lo que se denomina "lenguajes de servidor", que son interpretados para mostrar información.

Algunos lenguajes de servidor de los que se dispone en el mercado son ASP (Active Server

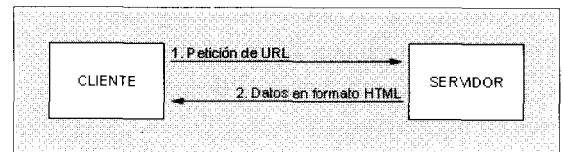


Figura 0.1.

Acceso a una página estática.

Pages), JSP (Java Server Pages), Cold Fusion o PHP (PHP Hypertext Processor), siendo este último uno de los más conocidos y utilizados, principalmente por el hecho de que puede ser utilizado en distintas plataformas (Unix, Linux, Windows, Mac) con el añadido de que un *script* desarrollado en un servidor concreto funcionará en cualquier otro sin prácticamente ninguna modificación. Es también un lenguaje *server-side* (todas las acciones que realiza tienen su efecto en el lado del servidor), y finalmente, porque se puede descargar de forma gratuita al ser su código abierto.

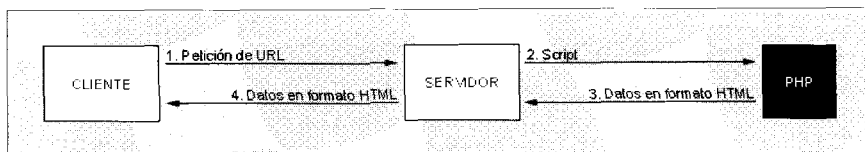


Figura 0.2.

Acceso a una página dinámica.

Podemos concluir entonces que los sitios Web actuales se han convertido en una suerte de "aplicación dinámica" que no se gestiona por aquellas personas que lo diseñaron, sino por los propios responsables del sitio, no creando nuevos documentos HTML añadidos al servidor vía FTP, sino mediante un conjunto de "paneles de control" que incluyen formularios que le permitirán la modificación, mediante el lenguaje de servidor utilizado, de los contenidos del sitio Web en cuestión en lo que se refiere tanto a textos como a archivos de imagen o multimedia.

Paralelamente al desarrollo de las bases de datos y los lenguajes de servidor, se producía el descubrimiento, auge y posterior "boom" de una de las herramientas de creación multimedia más conocidas estos días: Macromedia Flash. La riqueza sonora y visual que aportaba a los sitios Web era imposible de igualar por los aquel entonces "poco visualmente atractivos" sitios realizados en texto HTML.

Si en sus tres primeras versiones se utilizaba casi exclusivamente para desarrollar animaciones basadas en gigantescas líneas de tiempo repletas de fotogramas, a partir de la versión 4 se introdujo la posibilidad de crear contenidos que respondieran fielmente a la interacción del usuario con los mismos, gracias al entonces poco refinado "lenguaje" ActionScript. Sin embargo, en un primer momento, Flash se dio de bruces ante la necesidad de gestionar contenidos dinámicos, pues las posibilidades que ofrecía para tal tarea dejaban bastante que desear; tanto es así que para ello se utilizaba otro producto de la casa Macromedia, Macromedia Generator. La ventaja que ofrecía Flash, en cuanto a la comunicación visual se refiere, se perdía al ser los contenidos prácticamente inamovibles si no se disponía de los ficheros .fla originales para modificarlos.

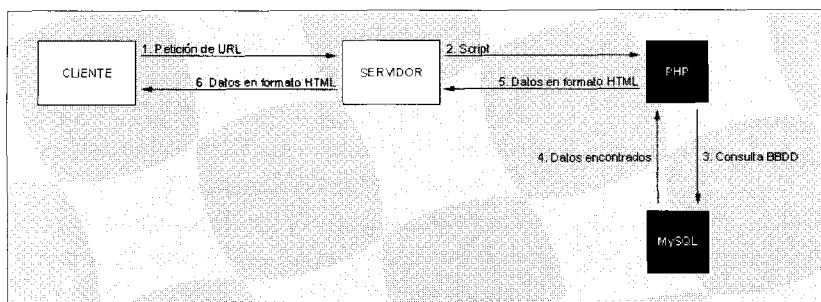


Figura 0.3.

Acceso a una página dinámica con información extraída de una base de datos.

La versión 5 de Flash contemplaba la evolución de ActionScript hacia un lenguaje de programación orientado a objetos, y aún más tras las mejoras introducidas en la versión MX de Flash, adaptando ActionScript a las especificaciones ECMA-262 (European Computer Manufacturing Association's ECMAScript Language Specification), con lo que en estos momentos disponemos de la posibilidad de comunicar y combinar nuestros contenidos Flash con .NET, servidores de *sockets* escritos en Java, JavaScript, XML y SQL2000, por ejemplo; no está mal para un pequeño *plug-in* que se descarga e instala en algo menos de un minuto en nuestro navegador, ¿no?

Hoy en día, combinar un software de desarrollo multimedia como puede ser Flash MX junto con un lenguaje de servidor realmente versátil, como pueda ser PHP, permite producir proyectos en los que tanto importa la riqueza visual de los mismos como la facilidad de mantenimiento y actualización por parte de quien contrata dicho sitio Web. En nuestro caso particular, y por poner un ejemplo, el sitio Web que realizamos a principios de 2003 para Carmen Heras (<http://www.carmenheras.org>) es uno de los trabajos de los que más satisfechos estamos, un proyecto donde absolutamente todos los contenidos que se muestran en el navegador dependen de la información que el cliente añade a la base de datos y el modo en cómo la dispone.

Aunque se cubren algunos fundamentos básicos de ambas disciplinas, este libro no pretende ser una guía de Flash ni un manual de PHP, sino que pretende mostrar la sencillez y las ventajas de la integración de ambas disciplinas de un modo eminentemente prácti-

co. De esta forma, aprenderá a utilizar Flash con datos dinámicos, aprenderá las nociones de PHP y MySQL necesarias para crear sus propias bases de datos y para añadir, modificar o eliminar contenidos de las mismas, y posteriormente aprenderá cómo trabajar conjuntamente con Flash y PHP, primero mediante ejemplos sencillos, y posteriormente mediante el desarrollo de un caso práctico que incluirá la realización de una agenda.

Para trabajar con este libro es recomendable que su sistema cumpla los siguientes requisitos:

- **Windows**

- Procesador 200 MHz o superior.
- Windows 98, SE, Windows Me, Windows NT 4, Windows 2000 o Windows XP.
- 64 MB de memoria RAM (recomendados 128 MB).

- **Macintosh**

- Mac OS 9.1 o superior, o bien OS X 10.1 o superior.
- 64 MB de memoria RAM (recomendados 128 MB).

Asimismo, necesitará disponer en su máquina de una versión de Flash MX para utilizar los ficheros incluidos en el CD-ROM. Si no dispone de una versión comercial puede utilizar una versión de prueba de treinta días que encontrará en el CD que acompaña a esta publicación. Además, también será necesario que instale y configure en su máquina el

servidor Web Apache, junto al intérprete de PHP y la base de datos MySQL. En el CD-ROM que acompaña a este libro encontrará todas las herramientas necesarias para poder comenzar a trabajar.

Si bien en la parte de PHP se explican las nociones básicas para trabajar partiendo desde cero, en lo que se refiere a Flash es conveniente que se posean conocimientos básicos en lo que se refiere al manejo de la herramienta y su entorno, creación de fotogramas clave, interpolaciones de movimiento y/o forma, herramientas de dibujo, etc.

Terminaremos esta pequeña introducción con la dirección Web en la cual podrá encontrar más información sobre este libro, sobre nosotros y sobre nuestro trabajo, así como un foro

en el que plantear dudas, aportar sugerencias, resolver problemas o simplemente mantener animadas conversaciones con quienes esto les escriben:

www.granatta.com/lib/flashphp

Y sin más preámbulos, vaya al comienzo del capítulo 1 para iniciar la lectura de este libro.

Daniel de la Cruz Heras
Granatta New Media Design
www.granatta.com

Carlos David Zumbado Rodríguez
Kadazuro
www.kadazuro.com

Junio de 2003

```
gotoAndPlay("capitulo_1");
```

Capítulo 1

Flash y contenidos dinámicos

Imagine que un día recibe un encargo para desarrollar, utilizando Flash, el sitio Web de una joyería y que, independientemente de los datos de contacto e información pertinente acerca de la misma, recibe usted de parte de los responsables un catálogo de quinientas joyas (con sus correspondientes quinientos textos descriptivos incluyendo características de las joyas, lugar de procedencia, precio, etc., y un par de fotos por cada una de las piezas) aguardando a ser digitalizado para dicho sitio Web. Tras una laboriosa tarea, consigue reducir cada uno de los textos a tan sólo 1 Kb, y el peso de cada una de las fotos a 10 Kb, tras lo que añade al documento .fla principal del sitio Web a desarrollar todo el material (figura 1.1), el cual, haciendo un cálculo aproximado de su tamaño ocupa:

$500 \text{ textos} * 1\text{Kb} = 500 \text{ Kb} \sim 0'5 \text{ Mb}$

$1000 \text{ fotos de joyas (500 joyas y dos fotos de cada una)} \times 10\text{Kb} = 10000 \text{ Kb} \sim 10 \text{ Mb}$

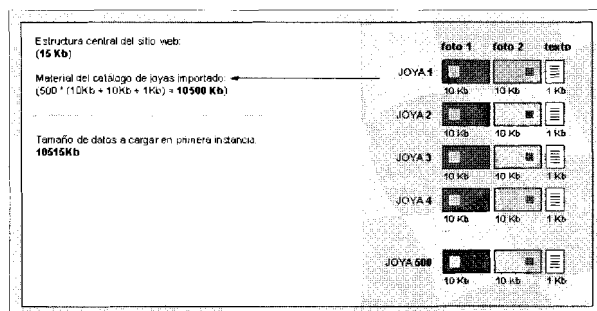


Figura 1.1.

Estructura del sitio Web con todo el material del catálogo importado.

Diez megas y medio para todo el material que los usuarios del sitio Web pueden desear visualizar, lo que huelga decir que es una auténtica barbaridad en tamaño. Piense además que puede que haya usuarios que tan

sólo deseen visualizar las joyas de plata y no las de oro, o que puede haber otros que sólo deseen información de las joyas que han sido fabricadas en un país concreto, por lo que ¿para qué vamos a almacenar toda la información en la película principal si puede que únicamente vayan a visitar un diez por ciento de la misma?

Sería por tanto más conveniente crear una estructura "central" en la cual mostrar la información, pero sin la obligatoriedad de almacenar todos los datos desde que entramos en el sitio Web, sino simplemente aquellos que los usuarios soliciten en un momento concreto (figura 1.2). De esta forma, pidiendo información sobre una joya cuyo nombre fuera, por ejemplo, *El sueño de Brasil*, tan sólo habríamos de solicitar la siguiente información:

1 texto x 1 Kb = **1Kb**

2 fotos de *El sueño de Brasil* x 10 Kb = **20 Kb**

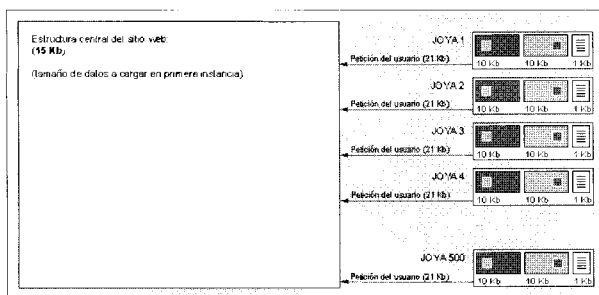


Figura 1.2.

Estructura del sitio Web cargando los contenidos de forma dinámica.

Lo que hace un total de 21 Kb por cada petición de información sobre cada una de las joyas, que es un peso bastante más llevadero para una conexión de Internet.

De alguna manera, y para bien o para mal (dependiendo de quien desarrollara los contenidos), Flash ofreció la posibilidad de llenar Internet de imagen y sonido; la ventaja era la de poder complementar (o incluso sustituir en determinados contextos) informaciones de tipo textual, pero también existía el inconveniente del peso de los documentos .swf que se visualizaban en los navegadores Web de los usuarios al incorporar imágenes con un mínimo de resolución o sonidos con un mínimo de calidad. La posibilidad de acceder a los contenidos (sean del tipo que sean) de forma individual y dinámica, y mediante requerimientos de los usuarios, solventa en gran medida el problema anteriormente citado: podemos crear una estructura básica sobre la que luego se irán montando el resto de contenidos como piezas de puzzle, de forma que, al final, el tratamiento de nuestro sitio Web para con los usuarios es mucho más eficiente, puesto que no hemos de hacerles esperar una eternidad para ofrecerles lo que buscan.

En este capítulo veremos, primeramente, algunas nociones básicas de uso de ActionScript para Flash MX, necesarias para que, posteriormente, comprenda todo lo necesario para aprender a manejar el modo en que Flash trabaja con los distintos contenidos dinámicos que puede importar:

- Documentos Flash (documentos de extensión .swf).
- Imágenes (documentos de extensión .jpg).
- Sonidos (documentos de extensión .mp3).
- Datos contenidos en ficheros de texto (documentos de extensión .txt).

- Datos contenidos en ficheros escritos en lenguajes de marcado XML (documentos de extensión .xml).



Nota: Asegúrese de tener instalada en su ordenador la última versión del player de Flash, con la que podrá acceder a todas las características mejoradas que ofrece con respecto a versiones antiguas. Puede descargarla en la dirección <http://www.macromedia.com/support/flash/downloads.html>, actualizando tanto el *plug-in* del navegador como el SWF Standalone Player (el reproductor que usa su ordenador para visualizar sus documentos .swf).

Nociones básicas de ActionScript para Flash MX

Tal y como dijimos en la introducción, se presuponen unos conocimientos mínimos de manejo de Flash MX para aprovechar al cien por cien los contenidos de este libro. Interpolaciones de forma, creación de símbolos e importación de archivos, por ejemplo, son habilidades que se dan ya por supuestas. Para trabajar con datos dinámicos es necesario el uso del lenguaje de programación de Flash MX: ActionScript. La actual versión de este lenguaje es la más completa de todas las versiones de Flash que han salido al mercado, y se ha intentado ajustar lo más posible a las especificaciones ECMA-262 (European Com-

puter Manufacturing Association's ECMAScript Language Specification), con lo que puede considerar ActionScript como un auténtico lenguaje de programación orientado a objetos. Tanto si no posee conocimientos de ActionScript como si son limitados, no se preocupe: este capítulo le servirá de rápido repaso de todo lo necesario para poder trabajar con datos dinámicos en Flash MX.

Trabajar en Modo Experto de edición

Trabajar con ActionScript es un acúmulo de dos facetas: el aprendizaje de la sintaxis del lenguaje como tal y, posteriormente, la aplicación de la misma para plasmar en la pantalla las ideas que usted tiene actualmente en la cabeza. Evidentemente, la primera faceta es la más simple: memorizar una serie de comandos e instrucciones que provocan acciones y eventos que se producen en sus películas Flash. La segunda faceta es un tanto más complicada; presupone el conocimiento de la sintaxis, pero también depende de la práctica que haya adquirido programando para que sus líneas de código sean más efectivas, más simples, más "pulidas", si quiere denominarlo así. Cuanta más práctica posea, más soltura adquirirá a la hora de teclear sus *scripts* de código.

Estos *scripts* vamos a escribirlos en la ventana de Acciones que posee Flash MX, la cual puede abrir pulsando **F9** cuando la aplicación esté abierta en su ordenador, o bien utilizando su ratón, como puede ver en las figuras 1.3 y 1.4, en las que, pulsando con el botón derecho del ratón sobre el fotograma 1 del documento .fla que tenemos abierto, seleccionamos la opción Acciones.

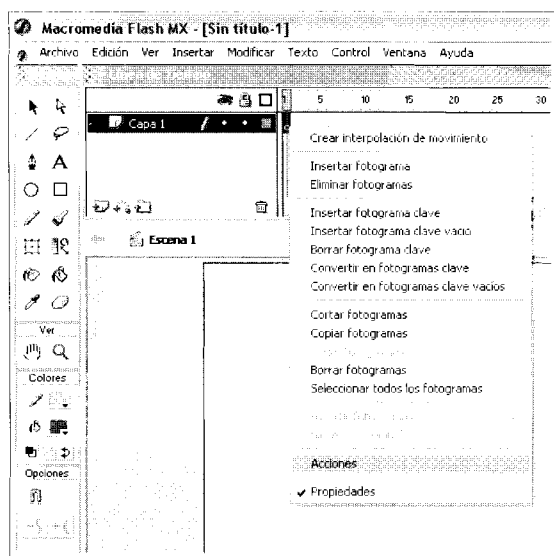


Figura 1.3.

Apertura de la ventana Acciones del fotograma seleccionado.

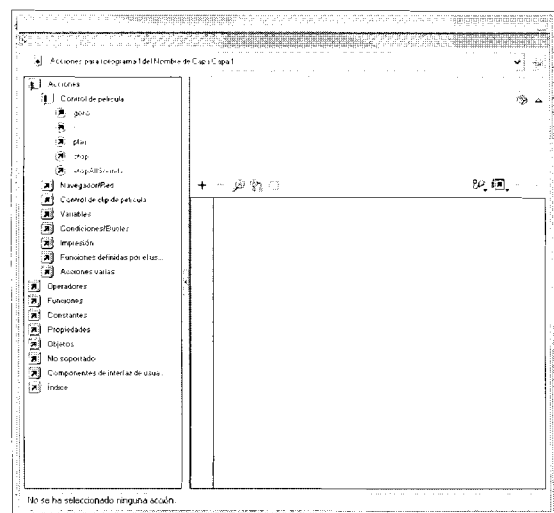


Figura 1.4.

Ventana Acciones en Modo de edición Normal.

Esta ventana dispone de dos formas de introducir código: el denominado Modo Normal, reminiscencia de la forma en que se introducía

código en la versión 4 de Flash y que consiste en recorrer los menús de la parte izquierda de la ventana de Acciones buscando la acción que deseamos añadir, para que, cuando esté elegida, añadir los parámetros en los cuadros de introducción de texto y desplegables que se mostrarán en la parte derecha de dicha ventana.

Este modo de edición, a primera vista, es más simple, pero tiene el inconveniente de que cuanto más complejas sean las líneas de código que ha de añadir, más tedioso será buscar cada una de las acciones en el menú de la parte izquierda. Para evitar esto, existe otro modo de edición, denominado Modo Experto, en el que usted puede teclear lo que desee en la parte derecha de la ventana. La primera sensación cuando se dispone de poca práctica programando en Flash es la de "caminar a ciegas". ¿Y si me equivoco? ¿Y si tecleo mal los comandos? Como ya hemos dicho antes, todo es cuestión de práctica, y si bien al principio puede que le cueste acostumbrarse al uso de este modo de edición, a la larga comprobará cómo el tiempo empleado para escribir sus *scripts* se reduce sensiblemente.

El modo de edición Normal es el que usted encontrará cuando abra por primera vez la ventana de Acciones en su versión de Flash MX. Para activar el Modo Experto despliegue con el botón izquierdo de su ratón el menú que hay en la parte superior izquierda de la ventana de Acciones y seleccione Modo Experto, tal y como se muestra en la figura 1.5.

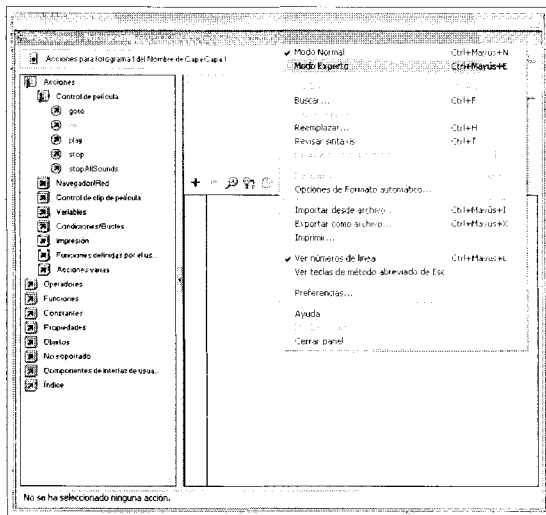


Figura 1.5.
Cambio de Modo de edición Normal a Modo Experto.

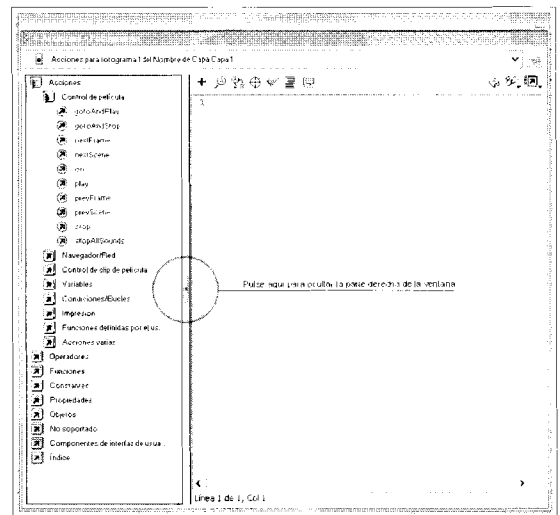


Figura 1.6.
Ventana Acciones en Modo de edición Experto.

A partir de este momento, ya puede escribir lo que desee en la parte derecha de la ventana de Acciones, e incluso puede pulsar la pequeña flecha que separa las dos partes de la ventana (figura 1.6) para que la zona de edición de texto ocupe la ventana completamente, como se puede ver en la figura 1.7. De esta forma maximizará la zona en la que puede teclear código sin modificar el tamaño de la ventana de Acciones.

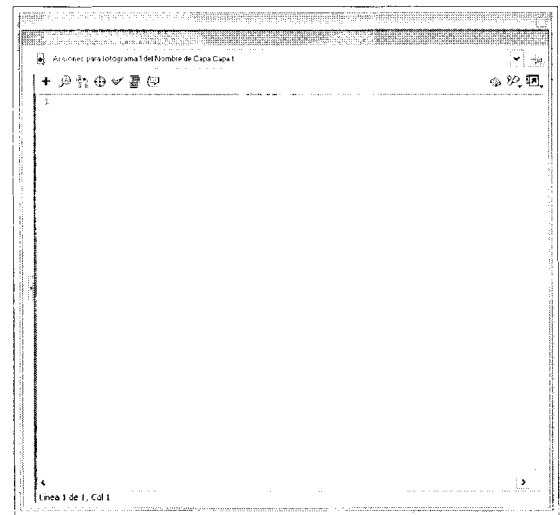


Figura 1.7.
Ventana Acciones en Modo de edición Experto ocupada completamente con la zona de edición de código.

Puede modificar los valores predeterminados para la ventana de edición de código mediante el Menú de Preferencias (Menú Edición> Preferencias>Editor ActionScript) para modificar los colores en los que se resaltarán palabras reservadas o comentarios, el sangrado de los bloques de texto, etc. (figura 1.8).

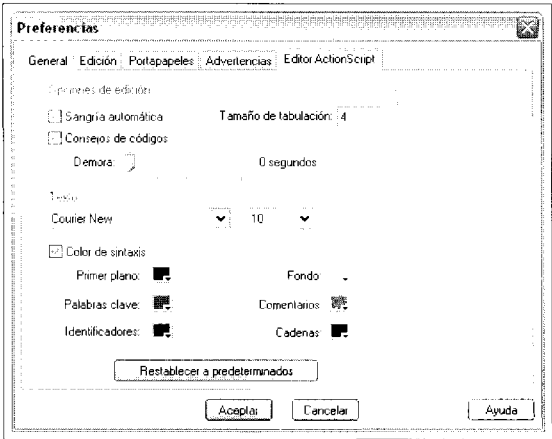


Figura 1.8.
Menú Preferencias del editor de ActionScript.

Variables

Cuando trabaje con Flash tendrá la necesidad de almacenar algunos de los valores que esté manejando para utilizarlos con posterioridad o definir acciones dependiendo de los mismos. Para ello, dispone de la posibilidad de utilizar variables. Una variable es esencialmente un contenedor de valores que almacena un contenido concreto en un momento dado, no pudiendo almacenar más de un valor simultáneamente. Los tipos de datos que pueden almacenar las variables en Flash puede observarlos en la tabla 1.1:

Tabla 1.1.

Tipos de datos que pueden almacenar las variables en Flash.

<i>Tipo de dato</i>	<i>Descripción</i>
Numérico	La variable almacena un número.
Texto	La variable almacena una cadena de texto.
Booleano	La variable almacena un valor booleano (verdadero o falso).

De esta forma, si tecleamos lo siguiente en alguno de los fotogramas de su película Flash:

```
quijote="En un lugar de la Mancha";
```

habremos creado una variable de nombre `quijote`, cuyo contenido es la cadena de texto (lo que se denota mediante colocar el texto entre comillas dobles) "En un lugar de la Mancha".

En cambio, si lo que tecleamos es:

```
mi_edad=26;
```

habremos creado una variable llamada `mi_edad` que almacena el valor numérico 26, como puede ver en la figura 1.9.

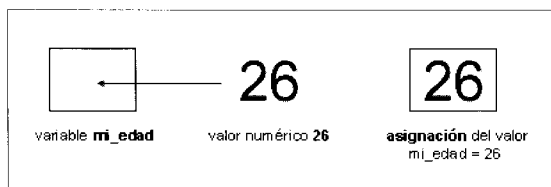


Figura 1.9.

Asignación del valor numérico 26 a la variable `mi_edad`.



Nota: El hecho de que una variable almacene un valor de un tipo determinado no supone que posteriormente no pueda almacenar datos de otro tipo distinto.

Para definir valores booleanos, en cambio, disponemos de dos posibilidades. La primera de ellas es la de asignar a la variable los valores `true` o `false`;

```
inicializado=true;
terminado=false;
```

Observe que asignar `"true"` a una variable es distinto de asignarle `true`, puesto que en el primer caso estamos asignando un valor textual, mientras que en el segundo asignamos un valor booleano.

La segunda forma de asignar valores booleanos es utilizando 0 ó 1 como números binarios, donde 0 es equivalente a `false` y 1 es equivalente a `true`, por lo que puede expresar las dos líneas anteriores de la siguiente forma:

```
inicializado=1;
terminado=0;
```

Una vez que ha almacenado valores en varias variables, puede operar con ellos o asignar los valores de unas a otras, por ejemplo:

```
mi_edad=26;
tu_edad=25;
edad_inicial=mi_edad;
nuestra_edad=mi_edad+tu_edad;
```

La variable `nuestra_edad` almacenará el valor resultante de la suma de las variables `mi_edad` y `tu_edad`, es decir, el valor numérico 51, mientras que la variable `edad_inicial` almacena el valor guardado en `mi_edad`, 26.

Al trabajar con variables, ha de asegurarse de que aquéllas con las que realice operaciones sean del mismo tipo, para prevenir situaciones como la que mostramos a continuación:

```
mi_edad=26;
tu_edad="25";
```

En este caso `mi_edad` es un valor de tipo numérico, mientras que `tu_edad` es una cadena de texto (puesto que el contenido se muestra entre comillas dobles), por lo que cuando realicemos la siguiente operación de adición

```
nuestra_edad=mi_edad+tu_edad;
```

el contenido de la variable `nuestra_edad` no será el valor numérico 51, sino la cadena de texto `"2625"`. Cuando trabajamos con cadenas de texto, el operador `"+"` se utiliza para concatenar las dos expresiones con las que se opera, por ejemplo:

```
silaba_1="Ka";
silaba_2="da";
silaba_3="zu";
silaba_4="ro";
nombre=silaba_1+ silaba_2+ silaba_3+
silaba_4;
```

de modo que en la variable `nombre` hemos guardado una cadena de texto resultado de concatenar las cuatro sílabas, resultando el texto "Kadazuro". En el ejemplo de las edades, se almacenaba en nuestra `edad` el valor 26, y posteriormente no se sumaba el valor numérico 25, sino que se concatenaba lo ya almacenado con la cadena de texto "25", resultando al final "2625" como valor almacenado en `nuestra_edad`.

Arrays

Un *array* es también una estructura contenedora de valores, pero, al contrario que las variables, que sólo podían almacenar un valor en cada instante, un *array* permite almacenar varios datos de forma simultánea. Piense en un *array* como uno de esos grandes ficheros que tiene varios cajones, en cada uno de los cuales hay información guardada sobre un tema concreto. El *array* dispone de varias posiciones en los que guarda datos (de igual forma a como lo hacen los cajones en el fichero) de cualquiera de los tres tipos que indicamos para las variables (puesto que, en esencia, cada una de las posiciones del *array* es una variable). Observe la figura 1.10 para comprobar cómo se guardan datos en las posiciones de un *array* de nombre `contenidos`:

`contenidos` guarda cinco valores, cada uno de los cuales es almacenado en las respectivas posiciones del *array*, que se numeran desde 0 hasta Número de elementos del *array*-1; al tener el *array* cinco posiciones, estas últimas se numeran desde 0 hasta 4.

contenidos				
10	"hola"	true	524	mi_edad
posición 0	posición 1	posición 2	posición 3	posición 4

Figura 1.10.

Valores almacenados en el array contenidos.

Para crear un *array* en Flash utilice una de las siguientes sintaxis:

```
contenidos=new Array(10, "hola",  
true, 524, mi_edad);
```

o bien:

```
contenidos=[10, "hola", true, 524,  
mi_edad];
```

El *array* `contenidos` almacena datos numéricos (10 ó 524), datos textuales ("hola") y datos booleanos (`true`), e incluso puede almacenar datos contenidos en otras variables (`mi_edad`).

Para acceder a cada uno de los valores guardados, basta con indicar el nombre del *array* junto con el índice de la posición de la que queremos conocer el dato, de modo que `contenidos[1]` contiene la cadena de texto "hola" y `contenidos[3]` almacena el valor 524.

Las posiciones del *array*, como hemos indicado anteriormente, funcionan como variables, así que puede alterar el contenido de las mismas si lo desea. Si, por ejemplo, teclee esto:

```
contenidos[0]="25";
```

la posición 0 del *array* reemplazará el valor 10 que guardaba por la cadena de texto "25".

La ventana de Salida

Cuando trabaje con ActionScript, rápidamente se dará cuenta de que es un lenguaje que podríamos considerar, en cierta forma, "visual": las acciones y comandos que usted teclee tendrán efecto sobre objetos que pueden cambiar de forma, color, posición, etc.; así, por ejemplo, si usted duplica la propiedad de anchura de un clip de película (*_width*), no le será necesario acudir a un depurador de código para comprobar si la anchura ha sido modificada, sino que le bastará con observar dicho clip de película en su pantalla.

No obstante, habrá casos en los que aquella propiedad o valor que deseamos "observar" es un tanto más abstracta que la anchura o altura de un clip de película; por ejemplo, a la hora de trabajar con clases e instancias de las mismas, por lo que necesitamos de alguna "herramienta" que nos permita seguirle los pasos a esos valores. Para ello, vamos a utilizar una de las ventanas que existen en el entorno de trabajo de Flash, la ventana de Salida (figura 1.11: Menú Ventana>Salida, o bien pulsando la tecla F2) que puede ver en la figura 1.12.

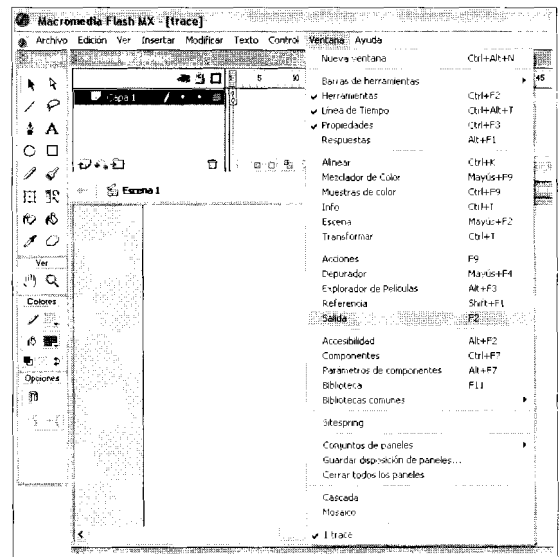


Figura 1.11.

Pasos para abrir la ventana de Salida.

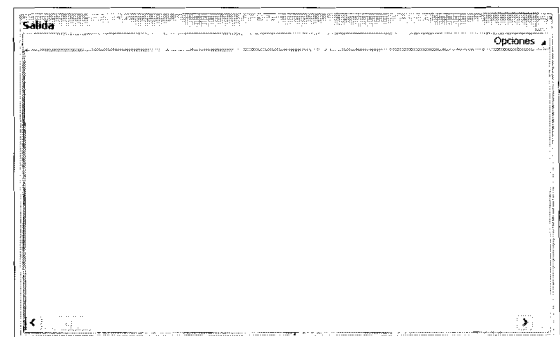


Figura 1.12.

Ventana de Salida.

Para mostrar contenidos en esta ventana será necesario que haga uso de la sentencia *trace*, cuya sintaxis es la siguiente:

```
trace("contenidos que desea mostrar  
en la ventana");
```

Por ejemplo, si desea mostrar en la ventana de Salida el texto "Esto es una prueba", no tiene más que crear un nuevo documento de Flash, abrir la ventana de Acciones y teclear en el primer fotograma:

```
trace("Esto es una prueba");
```

Como puede observar, ha mostrado en dicha ventana una cadena de texto, pero también puede mostrar el contenido de una o varias variables. Abra de nuevo el documento .fla que creó con anterioridad y escriba esto debajo:

```
a=16;
```

Hemos creado una variable llamada *a* y le hemos asignado el valor 16. Para mostrar el contenido de *a* en la ventana de Salida tan sólo hemos de añadir:

```
trace(a);
```

De igual forma, podemos mostrar el contenido de *arrays*. Añada esto debajo de sus líneas de código anteriores:

```
b=[30,20,"hola Caracola"];
trace("El contenido de la posición 0
es: "+b[0]);
trace("El contenido de la posición 1
es: "+b[1]);
trace("El contenido de la posición 2
es: "+b[2]);
```

En donde no sólo mostramos el contenido de cada una de las posiciones (de igual forma a como mostramos una variable), sino que concatenamos ese contenido con cadenas de texto en las que se especifica que "El contenido de la posición NÚMERO es", para que sepamos a qué posición estamos haciendo referencia. Puede encontrar el código que

hemos escrito en el documento material/cap1/fla/trace/trace.fl del CD.

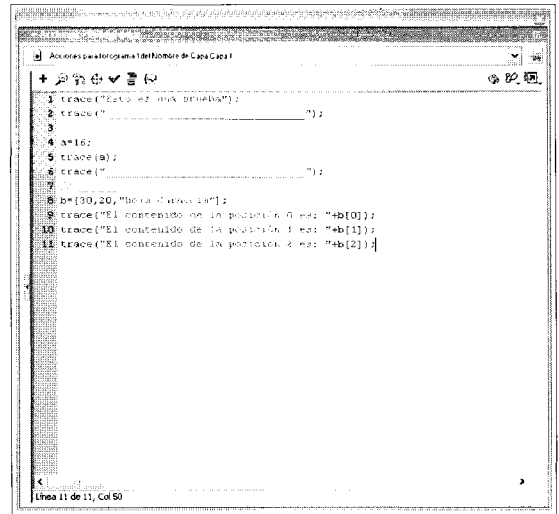


Figura 1.13.

Código tecleado en la ventana Acciones.

Si ahora reproduce su película desde el entorno de Flash (Menú Control>Probar Película, o bien pulsando **Control+Intro**) observará en su ventana de Salida algo similar a esto (figura 1.14):

```

Esto es una prueba
_____
16
_____
El contenido de la posición 0 es: 30
El contenido de la posición 1 es: 20
El contenido de la posición 2 es:
hola Caracola
  
```

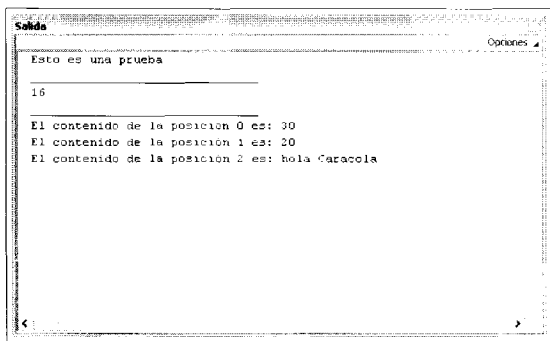


Figura 1.14.

Datos mostrados en la ventana de Salida a partir del código anterior.

Muchos de los contenidos de este libro se basarán en valores que, realmente, no tienen una representación en la pantalla (datos de un documento .xml externo, etc.), de forma que la ventana de **Salida** es una opción muy útil para observar si se han cargado o modificado correctamente, independientemente del uso que posteriormente tengamos pensado darle.



Nota: El uso de la acción `trace` es útil para el desarrollador, desde el punto de vista de mostrar valores en la ventana de **Salida** cuando está trabajando, pero una vez que se desea publicar el documento .swf a partir del .fla, es conveniente convertir aquellas líneas donde se usa `trace` a líneas de comentario, puesto que a fin de cuentas el visitante que verá el fichero de Flash no observará nunca en su navegador el efecto de los distintos `trace` que hayamos usado.

Funciones

Cuando esté trabajando con Flash se encontrará en numerosas ocasiones en la tesitura de que se repitan constantemente trozos de código. Puede utilizar funciones para escribir ese código una sola vez y reutilizarlo posteriormente en todas las ocasiones en que sea necesario invocando simplemente el nombre de la función.

Para declarar una función en Flash dispone de dos posibilidades:

```
nombre_funcion=function(parametros){
    acciones;
}
```

o bien:

```
function nombre_funcion(parametros){
    acciones;
}
```

Una vez declarada, para realizar una llamada a la función, basta con invocar su nombre junto con los parámetros necesarios (en el supuesto de que se hayan especificado al declarar la función). Si la función no tiene parámetros, se deja vacío el interior de los dos paréntesis al realizar la llamada:

```
nombre(parametros);
```

En la carpeta `material/cap1/fla/funciones` del CD encontrará varios documentos .fla en los que se trabaja con funciones. En el primero de ellos, `funciones1.fl`, hemos creado una función que nos permite calcular la suma de dos valores, 4 y 12:

```
function sumar(){
    resultado=4+12;
}
sumar();
trace(resultado);
```

lo que produce que se muestre en la ventana de **Salida** el valor almacenado en la variable `resultado`, 16. Esta función, `sumar`, no tiene parámetros, pero podemos modificar su declaración para calcular la suma de cualquier par de valores, como puede ver en el documento `funciones2.fla` del mismo directorio:

```
function sumar(a,b){
    resultado=a+b;
}
// -----
sumar(3,5);
trace(resultado);
trace("_____");
// -----
sumar(16,-3);
trace(resultado);
```

En la primera llamada a `sumar` se suman 3 y 5, con lo que `resultado` almacena el valor 8, mientras que en el segundo caso se suman 16 y -3, con lo que se almacena el valor 13, obteniendo lo siguiente en su ventana de **Salida** al reproducir la película:

```
8
-----
13
```

El utilizar como parámetros en la llamada a la función los valores 3 y 5 se suele expresar como "pasarle los valores 3 y 5 a la función `sumar`".

Retornar valores

El utilizar funciones no sólo permite "agrupar" un conjunto de acciones que se repite en nuestra película Flash, sino que también pueden retornar valores. Abra el documento `material/cap1/fla/funciones/funciones3.fla` del CD para ver cómo se ha redeclarado la función `sumar`:

```
function sumar(a,b){
    resultado=a+b;
    return resultado;
}
// -----
valor1=sumar(3,5);
trace(valor1);
trace("_____");
// -----
valor2=sumar(16,-3);
trace(valor2);
```

En este caso no se calcula el valor de la variable dentro de la función, sino que ésta calcula el resultado de la expresión `a+b` y lo asigna a una variable llamada `resultado`, que posteriormente es devuelta utilizando la acción `return`. Ese valor devuelto es el que se asigna a las variables `valor1` y `valor2` fuera de la función; así que, si reproduce su película, encontrará el siguiente contenido en su ventana de **Salida**:

```
8
-----
13
```

No obstante, no tiene por qué utilizar una variable que retorne el valor de la suma dentro de la función, sino que directamente puede declarar ésta de la siguiente forma:

```
function sumar(a,b){
    return a+b;
}
```

tal y como puede comprobar abriendo el documento `material/cap1/fla/funciones/funciones4.fla` del CD.

Variables locales

Las variables que hemos usado dentro de las funciones para operar con los valores que pasábamos por parámetro eran variables globales, podían ser accedidas desde la propia línea de tiempo y continuaban existiendo

aun cuando la función termine su actividad, lo que resulta en una ocupación innecesaria de memoria. Declarando las variables como locales dentro de la función, conseguiremos que, una vez que se ejecute la función, las variables utilizadas desaparezcan de nuestra película Flash. Para ello, escriba la palabra reservada `var` delante del nombre de la variable, tal y como puede ver si abre el documento `material/capl/fla/funciones/funciones5 fla` del CD, en el que encontrará el siguiente código en el fotograma 1 de la capa **Acciones**:

```
function sumar(a,b){
    var resultado;
    resultado=a+b;
    return resultado;
}
// -----
valor1=sumar(3,5);
trace(valor1);
trace("-----");
// -----
valor2=sumar(16,-3);
trace(valor2);
```

Si de nuevo reproduce su película, obtendrá la siguiente información en su ventana de Salida:

```
8
-----
13
```

tal y como ocurría en los ejemplos anteriores, pero con la salvedad, en este caso, de que el valor de `resultado` no es accesible desde la línea de tiempo principal, pues ha sido eliminada al concluirse la ejecución de la función.

delete

En ocasiones se encontrará en una situación en que no necesitará usar más una variable o función, de modo que puede usted utilizar la

sentencia `delete` para eliminar los que desee de su película Flash y evitar que ocupen memoria innecesariamente. La sintaxis para utilizar esta acción es:

```
delete nombre;
```

con lo que si dispone de una variable llamada `nuestra_edad`, puede usted eliminarla tecleando:

```
delete nuestra_edad;
```

Para ver otro ejemplo, abra el documento `material/capl/fla/delete/delete fla` del CD, en cuyo primer fotograma encontrará el siguiente código:

```
a=16;
trace(a);
delete a;
trace(a);

trace("-----");
// -----
// -----
b=[30,20,"hola Caracola"];
trace("El contenido de la posición 0
es: "+b[0]);
trace("El contenido de la posición 1
es: "+b[1]);
trace("El contenido de la posición 2
es: "+b[2]);
delete b;
trace(b[0]);
trace(b[1]);
trace(b[2]);

trace("-----");
// -----
// -----
saludo=function(){
    trace("hola");
}
trace(typeof(saludo));
delete saludo;
trace(typeof(saludo));
```

Si reproduce la película desde Flash, obtendrá estos resultados en la ventana de Salida:

```
16
undefined

-----
El contenido de la posición 0 es: 30
El contenido de la posición 1 es: 20
El contenido de la posición 2 es:
hola Caracola
undefined
undefined
undefined

-----
function
undefined
```

Como puede observar, tras cada uno de los valores mostrados correctamente mediante las llamadas a la sentencia `trace` se acompaña un valor de salida `undefined` (valor no definido o no existente), que se produce porque se realiza una llamada a `trace` con datos que ya no existen. Así, la variable `a`, que en un principio contiene el valor 16, desaparece de la película Flash al ser borrada, por lo que el resultado de ejecutar `trace(a)` retorna `undefined`, al igual que al intentar mostrar los contenidos del `array b` tras haberlo borrado o al intentar mostrar el tipo al que pertenece la función `saludo` una vez eliminada.

Jerarquías de clips

En el apartado 1.2.4, dedicado al uso de la ventana de Salida, comentábamos que ActionScript podía ser considerado un lenguaje "visual", cuyos efectos tenían repercusión sobre objetos de nuestra pantalla, generalmente clips de película. Es de vital importancia, por tanto, conocer el modo en que se accede a esos clips de película, a sus propiedades y métodos, y también cómo realizar llamadas de unos a otros.

Primeramente, habremos de acostumbrarnos a que, cada vez que dispongamos un clip de película en nuestro escenario, le asignemos un nombre de instancia. De esta forma podremos establecer comunicación con él desde otro clip o desde la línea de tiempo principal, puesto que, en caso de no darle nombre de instancia, el clip podría operar por sí mismo pero estaría "incomunicado".

Imaginemos ahora que disponemos en nuestra película Flash de un clip de película cuyo nombre de instancia al colocarlo en el escenario es `clip1`, dentro del cual existe otro de nombre `clip2`, que contiene un tercero llamado `clip3`.

`_root`

`_root` hace referencia a la línea de tiempo de la película principal, dentro de la cual están contenidos todos los clips de la misma. En este nuestro caso, el acceso a los distintos clips de que disponemos sería como sigue a continuación:

- Referenciar el `clip1` se realizaría mediante `_root.clip1`.
- Referenciar el `clip2` se realizaría mediante `_root.clip1.clip2`.
- Referenciar el `clip3` se realizaría mediante `_root.clip1.clip2.clip3`.

`this`

`this` hace referencia al clip de película en cuya línea de tiempo estemos trabajando; así, si tenemos declarada una variable llamada `valor` en `clip2`, podríamos acceder a ella de la siguiente forma:

- Acceder a la variable desde `_root` se realizaría mediante `_root.clip1.clip2.valor`.
- Acceder a la variable desde `clip1` se realizaría mediante `this.clip2.valor`.
- Acceder a la variable desde el propio `clip2` se realizaría mediante `this.valor`.

`_parent`

Hasta ahora hemos visto cómo recorrer la jerarquía de clips hacia delante, desde `_root` hasta `clip3`, pero también disponemos de la posibilidad de recorrerla hacia atrás. Para ello disponemos de la palabra reservada `_parent`, que hace referencia al clip inmediatamente anterior en la jerarquía de clips al que realiza la llamada. Por ejemplo:

- Acceder al contenido de la variable `valor` de `clip2` si estamos en `clip3` se realizaría mediante `_parent.valor`.
- Si en `clip1` hubiera declarada una variable llamada `letra`, acceder a ella desde `clip3` se realizaría mediante `_parent._parent.letra`. El primer `_parent` hace referencia al clip anterior en la jerarquía a `clip3` (`clip2`) y el segundo hace referencia al clip anterior en la jerarquía a `clip2`, es decir, `clip1`, dentro del cual está `letra`.

`_global`

Trabajando con Flash se encontrará en la situación de tener que utilizar valores o realizar llamadas a funciones declaradas dentro

de clips de película o en otras líneas de tiempo. Para ver un ejemplo, abra el documento `material/cap1/fla/global/no_global fla` del CD, en el que encontrará dos capas: *acciones* y *contenido*. En la primera de ellas hemos escrito el siguiente código:

```
a=16;

saludo=function(){
    trace("hola");
}
```

Como ya hemos visto previamente, estas líneas sirven para declarar una variable de nombre `a` y una función de nombre `saludo`. Si ahora deseáramos mostrar el valor de `a` en la ventana de Salida o realizar una llamada a la función desde la línea de tiempo principal, nos bastaría con teclear:

```
trace(a);
saludo();
```

Pero la cosa cambia si la llamada no se realiza desde la línea de tiempo principal. En la capa *contenido* encontrará un clip de película cuyo nombre de instancia es `cuadrado`, dentro del cual encontrará a su vez dos capas, una con el dibujo y otra llamada *acciones* en la que encontrará este código:

```
trace(_root.a);
trace("_____");
// _____
_root.saludo();
```

Puesto que estamos trabajando con una variable y una función que están en la línea de tiempo principal, hemos de indicarlo al hacer referencia a las mismas desde dentro del clip de película; de ahí el motivo de anteponer

`_root` al nombre. Si ahora reproduce su película, obtendrá este resultado en su ventana de **Salida**:

```
16
-----
hola
```

De donde se deduce que hemos accedido correctamente tanto al valor contenido en la variable como a la función de una forma relativamente sencilla. Sin embargo, si la variable estuviera declarada dentro de un clip de película, y dentro de éste hubiera otro con la declaración de la función, las llamadas y referencias serían bastante más confusas, ya que continuamente habríamos de recordar dónde se habían declarado. Por esta razón, puede declarar sus variables y/o funciones como globales anteponiendo la palabra reservada `_global` en la declaración, para, a partir de ese momento, poder hacer referencia desde cualquier línea de tiempo de la película Flash sin especificar una ruta o jerarquía de clips. Para ver un ejemplo, abra el documento `material/cap1/fla/global/global fla` del CD, que, esencialmente, es idéntico al documento `no_global fla` que vimos anteriormente. La diferencia estriba en que tanto la variable como la función se declaran como globales:

```
_global.a=16;

_global.saludo=function(){
    trace("hola");
}
```

Si ahora edita el clip de película cuyo nombre de instancia es `cuadrado`, podrá encontrar este código en la capa *acciones*:

```
trace(a);
trace("_____");
// _____
saludo();
```

Como puede observar, no se hace referencia a que `a` y `saludo` se han declarado en `_root`, porque al ser declaradas como globales no es necesario. Si ahora reproduce su película Flash, obtendrá esta información en la ventana de **Salida**, comprobando que puede acceder tanto al valor de la variable como a la función:

```
16
-----
hola
```



Nota: Cuando trabaje con variables o funciones globales asegúrese de que no crea ninguna otra variable o función con el mismo nombre dentro de otro clip de película, puesto que a la hora de intentar invocar estas últimas, Flash utilizaría las variables y funciones declaradas como globales.

Condiciones

Flash dispone de estructuras condicionales que permite que las películas que creamos actúen de una u otra forma dependiendo de que una determinada condición se cumpla o no:

```
si (condición) realizarAcciones();
```

O habilitando el que también se puedan desencadenar acciones alternativas si la condición no se cumple:

```
si (condición) realizarAcciones()
si no realizarOtrasAcciones();
```

Para expresar en Flash una estructura condicional utilizaremos la sentencia `if`:

```
if (condición){
    realizarAcciones();
}
```

o bien:

```
if (condición){
    realizarAcciones();
} else {
    realizarOtrasAcciones();
}
```

si lo que queremos es especificar acciones alternativas a realizar si no se cumple la condición.

Podemos incluso especificar varias condiciones que se comprueben y provocar que se ejecuten unas acciones u otras dependiendo de aquéllas:

```
if (condición1){
    realizarAcciones1();
} else if (condición2){
    realizarAcciones2();
} else if (condición3){
    realizarAcciones3();
} else {
    realizarOtrasAcciones();
}
```

En donde se comprueba si se cumple condición1. Si es así, se ejecutan las Acciones1, pero si no, se comprueba si se cumple condición2. Si se cumple, se ejecutan las Acciones2, pasando, si no se cumple, a comprobar condición3. Si ésta es cierta, se ejecutan las

Acciones3, y en caso contrario, se realizan las OtrasAcciones.

En la carpeta `material/cap1/fla/condicionales/` del CD encontrará varios documentos .fla con ejemplos de cómo manejar estructuras de condición en Flash. El primero de ellos, `if fla`, consta de un solo fotograma en el que encontrará este código:

```
a=26;

if (a>15){
    trace("A es mayor que 15");
}
```

El mensaje de que "A es mayor que 15" sólo se mostrará en la ventana de Salida si el valor de `a` es mayor que 15 (ésta es la condición en este caso). Si reproduce su película, podrá observar el mensaje, puesto que el valor de `a` es 26.

El segundo documento de que dispone en este directorio es `if_else fla`, que también consta de un único fotograma, con este código:

```
a=6;

if (a>15){
    trace("A es mayor que 15");
} else {
    trace("A es menor o igual que 15");
}
```

En este caso, especificamos que si `a` es mayor que 15, se muestre el mensaje "A es mayor que 15" en la ventana de Salida; y si no lo es, que se muestre "A es menor o igual que 15". Puesto que el valor de `a` es 6, el mensaje que se muestra es el segundo.

El último documento es `if_elseif fla`, en el que encontrará el siguiente código:

```
a=42;

if (a<=25){
    trace("A es menor o igual que 25");
} else if (a<=50){
    trace("A es menor o igual que 50 y mayor que 25");
} else if (a<=75){
    trace("A es menor o igual que 75 y mayor que 50");
} else {
    trace("A es mayor que 75");
}
```

En el que se establecen distintos rangos de valores en los que puede encontrarse el valor de `a`, dependiendo del cual se mostrará uno de los cuatro posibles mensajes en la ventana de **Salida**.

Bucles

Los bucles se utilizan para repetir un conjunto de instrucciones mientras se cumpla una determinada condición. Imagine que ha de abrir y cerrar cien cajones de un archivo de una biblioteca. Podría usted comenzar con el primero de ellos y terminar por el último:

```
abrir_Cajon_1();
cerrar_Cajon_1();
abrir_Cajon_2();
cerrar_Cajon_2();
abrir_Cajon_3();
cerrar_Cajon_3();
...
abrir_Cajon_100();
cerrar_Cajon_100();
```

Lo que resulta al final en doscientas líneas de código, aparte del hecho de que es tedioso escribir cien veces cada par de líneas, una por cada cajón. Utilizando bucles, puede escribir una sola vez el par de acciones

(`abrir_Cajon` y `cerrar_Cajon`) y será el propio bucle el que las ejecute tantas veces como usted establezca (en lo que se denominan *iteraciones* o *repeticiones* del bucle).

A continuación veremos brevemente dos tipos de bucle, `for` y `while`.

for

La manera en que se declara un bucle `for` es:

```
for (valor_inicial_de_la_variable,
    condición_de_continuidad,
    modificación_de_la_variable){
    realizarAcciones();
}
```

Abra el documento `material/cap1/fla/bucles/for/for fla` del CD, en el que encontrará, en el fotograma 1 de la capa *acciones*, el siguiente código:

```
for (i=1;i<=100;i++){
    trace("Abrir el cajón "+i);
    trace("Cerrar el cajón "+i);

    trace("_____");
}
```

Cuando este código se ejecute, se mostrarán en la ventana de **Salida** los mensajes de apertura y cierre de los distintos cajones especificados por la variable `i`, cuyo valor inicial es 1, y que se irá incrementando en una unidad en cada iteración mientras que su valor sea menor o igual que 100. De esta forma, si reproduce la película *Flash*, obtendrá estos datos en la ventana de **Salida**:

```
Abrir el cajón 1
Cerrar el cajón 1
-----
Abrir el cajón 2
Cerrar el cajón 2
-----
Abrir el cajón 3
Cerrar el cajón 3
```

```
-----  
Abrir el cajón 4  
Cerrar el cajón 4  
-----
```

```
Abrir el cajón 5  
Cerrar el cajón 5  
-----
```

```
...
```

```
-----  
Abrir el cajón 99  
Cerrar el cajón 99  
-----
```

```
Abrir el cajón 100  
Cerrar el cajón 100  
-----
```

En el momento en que el valor de *i* es superior a 100, el bucle deja de ejecutar las acciones, puesto que ya no se cumple la condición para ello (que el valor de *i* fuera menor o igual que 100).



Nota: *i++* es una expresión equivalente a *i=i+1*. Para más información, consulte los operadores de que dispone Flash MX.

La segunda forma de declarar un bucle *for* es:

```
for (variable in intervalo){  
    realizarAcciones();  
}
```

Para ver un ejemplo de su uso, abra el documento `material/cap1/fla/bucles/for/for_in.fla` del CD, en el que encontrará dos capas en el fotograma 1. La superior, llamada *contenidos*, contiene cuatro instancias (llamadas *cuadro1*, *cuadro2*, *cuadro3* y *cuadro4*) del clip de película de nombre *cuadro* guardado en la Biblioteca del documento *.fla*. En la inferior, llamada *acciones*, encontrará el siguiente código:

```
for (i in _root){  
    if  
    (typeof(_root[i])=="movieclip"){  
        trace(i+": "+_root[i]);  
  
        trace("_____");  
    }  
}
```

que sirve para que una variable llamada *i* recorra la película principal (*_root*) y compruebe si cada elemento que encuentra es o no un clip de película. Si lo encuentra, se muestra en la ventana de **Salida** un mensaje con el contenido en esa iteración de *i*, así como el nombre del clip. Al reproducir su película, obtendrá esta información en la ventana de **Salida**:

```
cuadro4: _level0.cuadro4
```

```
-----  
cuadro3: _level0.cuadro3
```

```
-----  
cuadro2: _level0.cuadro2
```

```
-----  
cuadro1: _level0.cuadro1  
-----
```

while

Un bucle de tipo *while* se construye conforme a la idea:

```
mientras (condición){  
    hacerAcciones();  
}
```

lo que en Flash se expresa mediante la sentencia *while*:

```
while (condicion){  
    hacerAcciones();  
}
```

Abra el documento `material/cap1/fla/bucles/while/while fla` del CD en cuyo primer fotograma encontrará el siguiente código:

```
i=1;

while(i<=100){
    trace("Abrir el cajón "+i);
    trace("Cerrar el cajón "+i);

    trace("_____");
    i++;
}
```

La primera línea sirve para inicializar el contenido de la variable `i`, puesto que no es posible inicializarla en la declaración del bucle tal y como ocurría en los bucles `for`. A continuación especificamos que, mientras que el valor de `i` sea menor o igual que 100, se ejecuten las acciones del interior del bucle, que son las mismas que declaramos anteriormente en el ejemplo de uso de `for`, con la salvedad de que hemos de añadir una acción que actualice el valor de `i`. Así, este bucle `while` ejecutará sus acciones mientras que el valor de `i` sea menor o igual que 100, dejando de ejecutarlas cuando alcance el valor 101. Si reproduce su película, puede observar, en la ventana de **Salida**, cómo la información es exactamente la misma que se mostraba utilizando un bucle `for`:

```
Abrir el cajón 1
Cerrar el cajón 1
```

```
-----
Abrir el cajón 2
Cerrar el cajón 2
```

```
-----
Abrir el cajón 3
Cerrar el cajón 3
```

```
-----
Abrir el cajón 4
Cerrar el cajón 4
```

```
-----
Abrir el cajón 5
Cerrar el cajón 5
```

```
-----
...
-----
Abrir el cajón 99
Cerrar el cajón 99
-----
Abrir el cajón 100
Cerrar el cajón 100
-----
```

Objetos

Los objetos son entidades existentes en nuestras películas Flash que tienen una serie de características que los definen (llamadas *atributos* del objeto), y una serie de actividades que pueden realizar (llamadas *métodos* del objeto).

Llamamos *Clase* al objeto en el que se definen las propiedades y los métodos. Posteriormente, utilizaremos *Instancias* de esa *Clase* en nuestros *scripts*, con lo que dichas instancias tendrán las propiedades y métodos de la clase a la que pertenecen, pudiendo usar las que Flash trae incorporadas o bien crear nuestras propias clases.

Para establecer una breve analogía entre los objetos del mundo real y aquéllos con los que trabaja Flash, podemos usar como ejemplo a un bebé. Un bebé tiene una serie de "atributos" que le definen: el color de sus ojos o el color de su pelo. También pueden realizar actividades, por ejemplo, llorar y comer, que vendrían a ser los "métodos" del objeto al que hemos llamado *bebé*.

Si ahora sabemos de las andanzas de dos bebés, Julián y Berceuse, podremos considerarles instancias de la clase *bebé*, y por ese motivo tendrán color de ojos y de pelo, de la misma manera en que podrán llorar y comer.

Los objetos en Flash funcionan de igual manera; por ejemplo, un clip de película dispuesto en su escenario es una instancia del objeto *MovieClip*. Por ello, la instancia tendrá propiedades como un nombre (`_name`), transparencia (`_alpha`), posición en x (`_x`) o en y (`_y`), etc., y métodos para ejecutar como `gotoAndStop()`, `gotoAndPlay()` o `duplicateMovieClip()`.

Además, disponemos de la posibilidad de "extender" los objetos en Flash para "enseñarles" a realizar determinadas tareas que en principio no pueden realizar. Por ejemplo, un clip de película dispone de la capacidad de reproducir su línea de tiempo mediante el método `play()`, pero no dispone de un método que le permita reproducir la línea de tiempo hacia atrás. Podemos crear ese método para la clase de los clips de película de igual manera que un bebé puede aprender a caminar y leer, con lo que a partir de ese momento todos los clips dispondrán de la posibilidad de reproducirse hacia atrás.

Clips de película creados dinámicamente. Dibujar mediante código

Una de las posibilidades más atractivas de las que dispone Flash MX es la de poder crear clips de película de forma dinámica, lo que le permitirá utilizarlos como "contenedores" para muy diversas tareas, desde cargar películas Flash externas o desde la propia Biblioteca, hasta utilizarlos para dibujar mediante código usando la conocida como Drawin' API de Flash MX. La sintaxis para crear un nuevo clip de película es:

```
createEmptyMovieClip("nombre_clip",profundidad);
```

Posteriormente veremos el uso de un clip creado dinámicamente para cargar contenidos externos, pero en este apartado vamos a centrarnos en un rápido repaso a las herramientas de dibujo mediante código de que dispone Flash MX.

Dibujar una línea

Una vez que hemos creado el clip de película, vamos a utilizarlo para dibujar dentro del mismo distintas líneas. Para ello, hemos de especificar el tipo de línea que queremos dibujar, según la siguiente sintaxis:

```
lineStyle(grosor, color, transparencia);
```

`grosor` es un número, en el que 0 significa "Línea muy fina", y de ahí en adelante se corresponde el número con el grueso del trazo. `color` es un número en formato hexadecimal comprendido dentro del rango 0x000000 (negro) y 0xFFFFFF (blanco) Por último, `transparencia` es un número comprendido entre 0 (completamente transparente) y 100 (completamente opaco).

Tras definir el tipo de línea, hemos de indicar un punto de "inicio" para comenzar a dibujar, lo que se especifica con el método `moveTo(x, y)`.

Finalmente, se especifica un punto de "llegada" para trazar la primera de las líneas, mediante el método `lineTo(x, y)`.

Abra el documento `material/cap1/fla/emptyclips_drawinAPI/poligono/dibujando.fla` del CD y observe el código escrito en el fotograma 1 de la capa *acciones*:

```
_root.createEmptyMovieClip("dibujo",1);
```

Se crea un nuevo clip de película.

```
with(dibujo){  
  lineStyle(2,0x666666,100);
```

Se define el grosor, color y transparencia de las líneas.

```
moveTo(100,100);
```

Se elige el punto (100, 100) como punto de partida.

```
  lineTo(300,100);  
}
```

Y para terminar, se traza una línea entre el punto de partida y el punto (300, 100).

Si ahora reproduce la película o abre el documento `dibujando.html` de la misma carpeta en su navegador, podrá observar cómo se ha dibujado una línea de color gris entre los puntos especificados.



Nota: Teclear

```
with(dibujo){  
  _x=10;  
}
```

es lo mismo que:

```
dibujo._x=10;
```

y sirve para acceder a los atributos del clip de película *dibujo*, en este caso a la posición en el eje X (`_x`).

Dibujar un polígono

De igual forma en que dibujamos una línea, podemos dibujar varias, siendo el comienzo de cada nueva línea el final de la anterior, siempre y cuando no indiquemos un nuevo punto de partida con `moveTo(x, y)`.

Observe el código del fotograma 1 de la capa *acciones* del documento `material/cap1/fla/emptyclips_drawinAPI/poligono/dibujando.fla` del CD:

```
_root.createEmptyMovieClip("dibujo",1);  
with(dibujo){  
  lineStyle(2,0x666666,100);  
  moveTo(100,100);  
  lineTo(300,100);  
  lineTo(300,150);  
  lineTo(250,250);  
  lineTo(120,240);  
  lineTo(100,100);  
}
```

Reproduzca la película o abra el documento `dibujando.html` del mismo directorio en su navegador para ver cómo se ha dibujado completamente un polígono de cinco lados en su pantalla.

Dibujar un polígono relleno

Además de crear polígonos también podemos establecer que tengan un relleno del color que nosotros elijamos, para lo que disponemos de los métodos `beginFill` y `endFill` del objeto *MovieClip*. La sintaxis de `beginFill` es:

```
nombre_clip.beginFill(color_relleno,  
  transparencia);
```

y la de `endFill` es:

```
endFill();
```

Todos los dibujos de línea que se encuentren entre `beginFill` y `endFill` provocarán que se rellene del color elegido aquellas superficies que se hayan cerrado. Observe el código del fotograma 1 de la capa *acciones* del documento `material/cap1/fla/emptyclips_drawinAPI/poligono_relleno/dibujando.fla` del CD:

```

_root.createEmptyMovieClip("dibujo",1);
with(dibujo){
    lineStyle(2,0x666666,100);
    beginFill(0xFFFFFF,100);
    moveTo(100,100);
    lineTo(300,100);
    lineTo(300,150);
    lineTo(250,250);
    lineTo(120,240);
    lineTo(100,100);
    endFill();
}

```

Se dibuja el mismo polígono del ejemplo anterior, pero al estar tanto `moveTo` como los distintos `lineTo` entre `beginFill` y `endFill` se rellena de color blanco la figura que hemos dibujado, lo que puede comprobar reproduciendo la película Flash o abriendo el documento `dibujando.html` de la misma carpeta en su navegador.

Campos de texto creados dinámicamente. Formatos de texto

Otra de las posibilidades que ofrece Flash MX es la de crear campos de texto de forma dinámica mediante código. La sintaxis para realizar esta tarea es la siguiente:

```

createTextField("nombre_campo",
profundidad, posición_x, posición_y,
anchura, altura);

```

Observe la figura 1.15 para entender qué representa cada uno de los parámetros.

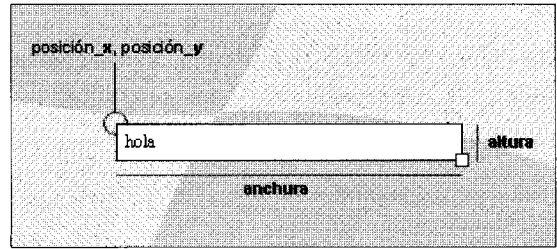


Figura 1.15.

Creación del campo de texto de forma dinámica.

Una vez creado, puede introducir texto en el atributo `text` de dicho campo para visualizarlo en pantalla; por ejemplo:

```

nombre_campo.text="hola";

```

En estos momentos, el texto carece de formato, por lo que puede crear un objeto de tipo *TextFormat* (Formato de Texto) que complementa al objeto *TextField*, de forma que pueda definir tanto el tipo de fuente, el tamaño, el sangrado de párrafo, etc., como puede observar en la figura 1.16. La sintaxis de creación de un objeto *TextFormat* es:

```

nuevo_formato=new TextFormat();

```

después de lo cual puede definir diversos atributos como `font`, `size`, `blockIndent`, `color`, `italic`, `bold`, etc.

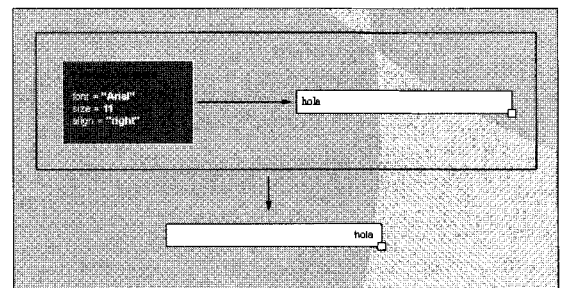


Figura 1.16.

Aplicación del formato de texto al campo de texto.

Cuando haya creado dicho formato y asignado valor a los atributos que haya creído convenientes, puede aplicar dicho formato al campo de texto siguiendo la sintaxis:

```
nombre_campo.setTextFormat(nuevo_formato);
```

Abra el documento `material/cap1/fla/textfields_textformat/textfield fla` del CD y vaya al fotograma 1 de la capa *acciones*, en la que encontrará el siguiente código:

```
formato=new TextFormat();
with(formato){
    font="Arial";
    size="11";
    align="right";
}
```

Definimos un formato de texto en el que se utilizará la tipografía *Arial*, a un tamaño de 11 puntos y con el texto alineado a la derecha. Posteriormente creamos un nuevo campo de texto:

```
_root.createTextField("nuevo_texto",1,50,
50,230,250);
```

Y ahora damos valor a los atributos que deseemos, por ejemplo, que tenga borde y fondo, así como color para cada uno de ellos. Establecemos también que el campo admita más de una línea y que el color del texto sea blanco:

```
with(nuevo_texto){
    border=1;
    borderColor="0x333333";
    background=1;
    backgroundColor="0x888888";
```

```
multiline=1;
wordWrap=1;
textColor="0xFFFFFFFF";
text="Este es un campo de texto
creado de forma dinámica. Tiene
borde y fondo, así como colores para
cada uno de ellos y utiliza una
Arial de 11 puntos para los textos
que aparezcan aquí, alineados además
a la derecha, lo que se define en el
formato de texto correspondiente."
}
```

Por último, asignamos el formato al campo de texto:

```
nuevo_texto.setTextFormat(formato);
```

con lo que si ahora reproduce la película o abre el documento `textfield.html` de la misma carpeta en su navegador, podrá observar cómo se ha creado el campo de texto de acuerdo a cómo nosotros hemos tecleado el código.



Nota: Las nociones que hemos visto sobre ActionScript en este apartado del libro deben serle de utilidad para poder continuar a partir de ahora y combinar Flash con contenidos dinámicos, pero si desea profundizar en el aprendizaje de este lenguaje de programación puede acudir al capítulo de recursos de este libro (capítulo 8) en el que encontrará enlaces a sitios Web y/o libros que pueden serle de utilidad para tal propósito.

Carga dinámica de documentos Flash (.swf)

La carga dinámica de documentos Flash le permite no almacenar toda la información de su sitio en un solo documento .fla (lo que acarrearía un incremento del peso de la animación final), sino que puede disponer de un documento raíz en el que se vayan cargando otros externos con información, en base a peticiones del usuario, tal y como vimos en el ejemplo del comienzo de este capítulo 1. La sintaxis para cargar un documento .swf externo desde nuestra película Flash es la siguiente:

```
clip_película.loadMovie("nombre_clip_externo.swf");
```

Para ver un ejemplo, vaya a la carpeta `material/cap1/fla/contenidos_dinamicos/swf/clip_escenario/` del CD, en la que encontrará un documento llamado `carga_swf.fla` (documento principal) y

otros tres llamados `externo_1.fla`, `externo_2.fla` y `externo_3.fla` (información externa). Abra cualquiera de estos tres últimos (los tres están contruidos de la misma forma) un documento .fla de 310x200 píxeles con la información almacenada en el fotograma 1 de la capa *contenidos*. Abra ahora el documento `carga_swf.fla`, en cuyo primer fotograma encontrará, en la capa *botones*, tres instancias del clip de película *botón* almacenado en la Biblioteca del documento .fla. Cada una de estas instancias ha recibido los nombres *boton1*, *boton2* y *boton3*, respectivamente. En la capa *clip vacío*, encontrará una instancia (de nombre *contenedor*) del clip vacío de la Biblioteca. Éste es el clip de película que utilizaremos para cargar los contenidos externos, para lo que hemos añadido el siguiente código a la capa *acciones*, en el que especificamos que cuando pulsemos cada uno de los botones se cargue en el clip *contenedor* el documento externo correspondiente:

```
boton1.onRelease=function() {  
    contenedor.loadMovie("externo_1.swf");  
}  
  
boton2.onRelease=function() {  
    contenedor.loadMovie("externo_2.swf");  
}  
  
boton3.onRelease=function() {  
    contenedor.loadMovie("externo_3.swf");  
}
```

Reproduzca la película o abra el documento `carga_swf.html` de la misma carpeta en su navegador y pulse los distintos botones para observar cómo se cargan los documentos .swf externos dentro del principal. Vaya ahora a la carpeta `material/cap1/fla/contenidos_dinamicos/swf/clip_codigo/` del CD, en la que encontrará

documentos con los mismos nombres que en el ejemplo anterior con una única salvedad: en el documento `carga_swf.fl`a hemos eliminado la capa *clip vacío* y hemos añadido estas líneas a la capa *acciones*:

```
_root.createEmptyMovieClip("contenedor",10);
contenedor._x=contenedor._y=70;
```

Con lo que, en vez de añadir un clip vacío al escenario de forma manual, lo hemos creado dinámicamente mediante código, para posteriormente recolocar en la posición (70,70). Si reproduce la película o abre `carga_swf.html` en su navegador, podrá observar, al pulsar los botones, cómo la funcionalidad sigue siendo la misma.

Carga dinámica de imágenes (.jpg)

De igual forma en que se cargan documentos .swf externos, podemos cargar archivos de imagen en formato JPEG. Abra la carpeta `material/cap1/fla/` contenidos_dinamicos/jpg/ del CD, en la que encontrará tres imágenes en formato JPEG y un documento llamado `carga_jpg.fl`a. Si abre éste último, podrá observar cómo el contenido del mismo es exactamente igual que el de `material/cap1/fla/contenidos_dinamicos/swf/clip_codigo/carga_swf.fl`a, con la salvedad de que encontrará este código en la capa *acciones*:

```
_root.createEmptyMovieClip("contenedor",10);
contenedor._x=contenedor._y=75;

boton1.onRelease=function(){
    contenedor.loadMovie("kadazuro.jpg");
}

boton2.onRelease=function(){
    contenedor.loadMovie("granatta.jpg");
}

boton3.onRelease=function(){
    contenedor.loadMovie("los_dos.jpg");
}
```

Tras crear un nuevo clip vacío que haga las veces de contenedor de la información externa, se asignan a los clips de película de la parte inferior las acciones necesarias para cargar las distintas imágenes disponibles en el directorio. Reproduzca la película o abra el documento `carga_jpg.html` de la misma carpeta en su navegador para ver las imágenes pulsando los distintos botones.



Nota: Asegúrese de que sus documentos .jpg son JPEG NO progresivos. En caso de ser progresivos, no podrá importarlos dinámicamente a su película Flash.

Carga dinámica de sonidos (.mp3)

Una de las características más atractivas que siempre ha presentado Flash es la de poder añadir contenidos sonoros que complementarían las informaciones textuales o visuales que mostraban los trabajos realizados. El gran inconveniente era el del peso de los archivos con los que se trabajaba (.wav en PC, .aiff en Macintosh), que incrementaban enormemente el número de Kb de los documentos .swf que finalmente se publicaban en la red. La versión 4 de Flash seguía trabajando con este tipo de archivos, pero permitía exportarlos como MP3, con lo que se consiguió que el peso se redujera considerablemente. Incluso en la versión 5 se pudo, por fin, importar documentos .mp3 a la Biblioteca de las películas Flash, pero aun así, era muy común alojar el sonido en un .fla externo e importar el .swf generado de éste mediante loadMovie. La versión MX de Flash permite importar documentos .mp3 de forma dinámica, lo que aligera en gran medida el tamaño de los documentos a subir al servidor. Como ejemplo de carga dinámica de un archivo .mp3, hemos utilizado una composición llamada "Nopalitos asados" de nuestro buen amigo Rolf Ruiz (<http://www.alesys.net>), que no sólo es un excelente desarrollador de Flash, sino también un mag-

nífico guitarrista; no en vano procede académicamente de estudios relacionados con la música (si tiene interés o le agradan sus composiciones, puede visitar algunos de los experimentos que ha realizado combinando Flash y música en <http://www.alesys.net/insp/su03.html>).

En la carpeta `material/cap1/fla/` contenidos_dinamicos/mp3/no_streaming/ del CD encontrará un documento .mp3 correspondiente a la canción y otro llamado `carga_mp3.fla`. Abra éste, en el que encontrará tres clips de película colocados en la capa llamada *botones*, con los nombres de instancia *cargar*, *reproducir* y *detener* asignados. El primero de ellos importará el archivo de sonido a la película Flash, mientras que los otros dos, como su propio nombre indica, sirven para reproducir y detener el sonido respectivamente. En la capa *texto* encontrará dos campos de texto con formato HTML, *titulo* (que se usa para avisar de que el sonido se ha cargado correctamente) y *status* (que sirve para indicar si el sonido se está reproduciendo o si está detenido). Por último, en la capa *acciones* encontrará el siguiente código (que explicamos brevemente) para importar el sonido y poder reproducirlo/detenerlo:

```
nombre_cancion="1.Alesys.net-  
NopalitosAsados.mp3";  
playing=0;
```

Definimos una variable con el nombre del fichero .mp3, ya que al ser un tanto largo, en caso de que se desee modificar su nombre en la carpeta del directorio, nos bastará con modificar el contenido de la variable para actualizar nuestro documento .fla, en vez de tener que buscar todas las apariciones del

nombre de la canción en el código. Posteriormente creamos una variable que llamaremos `playing`, cuyo contenido es booleano, donde `true` significa que el sonido se está reproduciendo y `false` significa que está detenido. La razón de ser de comprobar si el sonido está en reproducción es porque si lo está, y volvemos a pulsar el botón **PLAY**, comenzará a reproducirse el sonido de nuevo sin interrumpirse el que estaba sonando, "montándose" uno encima del otro.

```
status.htmlText="<p  
align=\"center\">Pulse el botón  
<b>CARGAR</b> para importar el  
sonido a esta película</p>";
```

A continuación establecemos un texto para `status` (añadiendo el contenido al atributo `htmlText` del campo de texto) y creamos un nuevo objeto de sonido, al que llamaremos `nopalitos`.

```
nopalitos=new Sound(this);  
reproducir.enabled=detener.enabled=0;  
reproducir._alpha=detener._alpha=30;
```

Establecemos el funcionamiento de los botones de la siguiente forma: mientras que el sonido no esté cargado, los botones **reproducir** y **detener** estarán inactivos. Una vez cargado, se deshabilitará el funcionamiento del botón **cargar** y se habilitarán los otros dos para manejar el sonido.

```
nopalitos.onSoundComplete=function(){  
    nopalitos.start(0,1);  
}
```

`onSoundComplete` es uno de los eventos a los que reacciona el objeto *Sound*. Las acciones declaradas dentro de la función se ejecutan una vez que el sonido ha terminado su reproducción, de modo que si establecemos que cuando el sonido termina de sonar vuelva a reproducirse, conseguiremos un efecto de *loop* para la canción:

```
nopalitos.onLoad=function(){  
    reproducir.enabled=detener.enabled=1;  
    reproducir._alpha=detener._alpha=100;  
  
    cargar.enabled=0;  
    cargar._alpha=30;  
  
    titulo.htmlText="<p align=\"center\"><b>"+nombre_cancion+"</b> se ha cargado  
correctamente</p>";  
    status.htmlText="<p align=\"center\">Pulse el botón <b>PLAY</b> para comenzar  
la reproducción del sonido</p>";  
    nopalitos.setVolume(100);  
}
```

`onLoad` es otro evento a los que reacciona el objeto *Sound*, por el que las acciones declaradas en el interior de la función se ejecutan cuando el sonido se ha cargado por completo. En este caso, las acciones consisten en habilitar los botones **reproducir** y **detener** e inhabilitar el botón **cargar**, además de hacer

referencia en los campos de texto al hecho de que el sonido se ha importado correctamente y que puede pulsar el botón **reproducir** para que la canción comience a sonar. Finalmente, se establece que el volumen del sonido sea 100, mediante una llamada al método `setVolume` del objeto *Sound*:

```
objeto_sonido.setVolume(número);
```

donde el parámetro `número` es un valor comprendido entre 0 y 100.

```
cargar.onRelease=function(){
    status.htmlText="<p align=\"center\">Importando el archivo de sonido
<b>"+nombre_cancion+"</b></p>";
    nopalitos.loadSound(nombre_cancion,0);
}
```

Definimos las acciones para cargar el sonido asociándolas al clip de película *cargar*. Aparte de indicar en el campo de texto *status* que estamos cargando la canción, usamos para comenzar la descarga el método `loadSound` del objeto *Sound*, cuya sintaxis es la siguiente:

```
objeto_sonido.loadSound("nombre_sonido.mp3", streaming);
```

donde el parámetro `streaming` hace referencia a si deseamos que el sonido se cargue en nuestra película Flash o deseamos que se escuche a medida que se descarga. En nuestro ejemplo, hemos elegido que el valor de ese parámetro sea 0 (`false`), con lo que se descargará pero no comenzará a sonar hasta que pulsemos el botón **Reproducir**.

```
reproducir.onRelease=function(){
    if (!playing){
        status.htmlText="<p align=\"center\">Pulse el botón <b>STOP</b> para
detener la reproducción del sonido</p>";
        nopalitos.start(0,1);
        playing=1;
    }
}
```

Las acciones para reproducir el sonido las hemos asociado al clip *reproducir*, y consisten en que se compruebe si el sonido está actualmente detenido o no (mediante una comprobación de la variable `playing`). En caso de que esté detenido, se establece la variable `playing` con el valor `true` (puesto que empezará su reproducción), y establecemos que comience a reproducirse mediante una llamada al método `start` del objeto *Sound*, cuya sintaxis es:

```
objeto_sonido.start(retraso_milisegundos,
repeticiones);
```

donde `retraso_milisegundos` es el número de milisegundos que tardamos en comenzar a escucharlo y `repeticiones` es el número de veces que deseamos que se reproduzca. Puesto que hemos establecido que cuando termine de sonar la canción vuelva a comenzar, no necesitamos más que establecer una sola repetición de la reproducción.

```
detener.onRelease=function() {
    if (playing) {
        status.htmlText="<p
            align=\"center\">Pulse el botón
            <b>PLAY</b> para comenzar la
            reproducción del sonido</p>";
        nopalitos.stop();
        playing=0;
    }
}
```

Las acciones para detener la reproducción del sonido se asocian al clip *detener*, y básicamente son las inversas a las del clip *reproducir*, estableciendo el valor de `playing` como `false` y deteniendo el sonido usando el método `stop` del objeto *Sound*, cuya sintaxis es:

```
objeto_sonido.stop();
```

Si ahora reproduce su película Flash o bien abre el documento `carga_mp3.html` de la misma carpeta en su navegador, puede pulsar

el botón **CARGAR** para importar el sonido y posteriormente **PLAY** o **STOP** para reproducirlo o detenerlo.

Vaya ahora a la carpeta `material/cap1/fla/contenidos_dinamicos/mp3/streaming/` del CD, donde encontrará los mismos documentos que en la carpeta anterior; abra el documento `carga_mp3.flay` vaya al fotograma 1 de la capa *acciones*. Esencialmente el código es el mismo, excepto aquellas partes en donde vamos a indicar que, a medida que se carga la canción, comience su reproducción en nuestra película Flash. Por ello, la única línea que cambia (a excepción de los mensajes en donde se actualiza el valor del campo de texto `status`) es la siguiente, dentro de las acciones definidas para el clip de película *cargar*:

```
cargar.onRelease=function() {
    status.htmlText="<p
        align=\"center\">Pulse el botón
        <b>STOP</b> para detener la
        reproducción del sonido</p>"
        nopalitos.loadSound(nombre_
        cancion,1);
        playing=1;
    }
```

La llamada al método `loadSound` se realiza con un valor `true` en el parámetro utilizado para indicar *streaming*, por lo que cuando pruebe su película o abra el documento `carga_mp3.html` de la misma carpeta en su navegador, podrá comprobar que el pulsar el botón **CARGAR** implica también el comienzo de la reproducción del sonido.

Carga dinámica de datos almacenados en ficheros de texto (.txt)

Cuando hemos de disponer grandes cantidades de información textual en nuestros trabajos es conveniente, al igual que con el resto de contenidos, almacenarla de forma externa e importarla en el momento en que el usuario lo solicita. Uno de los métodos más utilizados es el de guardar la información en forma de variables en documentos de texto de extensión .txt. El modo en que se genera contenido para las variables sigue la siguiente sintaxis:

```
&nombre_variable1=contenido de la
variable1&nombre_variable2=contenido
de la variable2, etc.
```

Se antepone al nombre de la variable el carácter &, y se le asigna contenido mediante el carácter =. En caso de haber más de una variable, basta con añadirla a continuación de la anterior, siempre indicando mediante & que estamos declarando una variable nueva.

Como ejemplo, vaya a la carpeta `material/cap1/fla/contenidos_dinamicos/txt/simple/` del CD y abra el documento `datos_autores.txt`, en el que encontrará los siguientes contenidos referentes a los autores de este libro:

```
&autor1=Daniel de la Cruz
Heras&pais1=España&mail1=mind@granatta.com&url1=
www.granatta.com&autor2=Carlos David
Zumbado Rodriguez&pais2=Costa
Rica&mail2=nadie@kadazuro.com&url2=
www.kadazuro.com
```

Ocho variables, cada una de ellas con su nombre antecedido por el carácter &. Muy importante es que a la hora de salvar este

documento .txt no lo haga con formato ANSI, sino como Unicode (Menú Archivo>Guardar como...), tal y como puede verse en la figura 1.17, para conseguir que se visualicen correctamente aquellos caracteres propios del alfabeto español (tildes, "ñ", diéresis, "í", "¿", etc.), tanto en PC como en Macintosh.

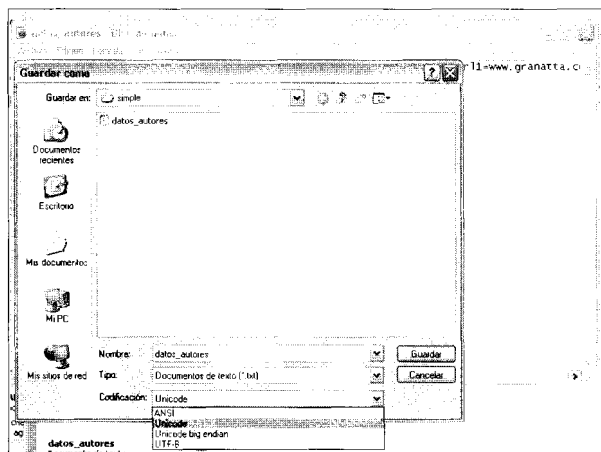


Figura 1.17.

Guardar el documento .txt con formato Unicode.

Ahora vamos a ver cómo Flash importa los datos que hemos almacenado en nuestro documento .txt, para lo que vamos a utilizar uno de los objetos de que dispone Flash MX denominado `LoadVars`. Abra el documento `carga_txt fla` que encontrará en la misma carpeta. En el fotograma 1 de la capa *acciones* se encuentra el siguiente código:

```
datos_autores=new LoadVars();
```

Primeramente se declara un nuevo objeto *LoadVars*, al que llamaremos `datos_autores`, que vamos a utilizar para almacenar los datos que aún están en el documento `datos_autores.txt`. Estos datos, que están

en el formato que hemos indicado antes, se guardarán dentro del objeto *LoadVars* que creemos de la forma en que puede verse en la figura 1.18; es decir, si en el documento .txt hemos declarado una variable llamada *saludo* de la forma *&saludo=Hola!*, en el objeto *LoadVars* en el que se importe el .txt se creará una variable con ese nombre (*saludo*), por lo que para acceder a su contenido nos bastará con referenciarla mediante *nombre_objeto_LoadVars.saludo*.

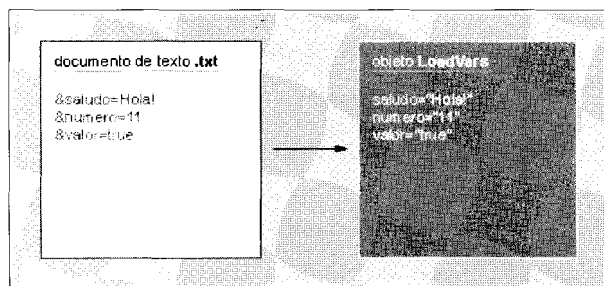


Figura 1.18.

Almacenamiento de las variables del documento .txt en el objeto LoadVars.



Nota: Los contenidos de las variables, al importarse dentro del objeto *LoadVars*, lo harán como cadenas de texto, por lo que si queremos usarlas como valores numéricos o booleanos habremos de convertirlos previamente.

La línea inmediatamente anterior sirve para realizar la llamada al documento .txt del que queremos extraer los datos usando el método *load* del objeto *LoadVars*. Una vez que se ha producido la carga (exitosa o no) del contenido externo, se "disparan" las acciones contenidas en el evento *onLoad* del objeto, que dispone de un parámetro (*exito*) cuyo valor es *true* si la carga se realizó correctamente, y *false* si se produjo algún error y la carga no pudo realizarse. En este último caso se muestra, por la ventana de Salida, un mensaje avisando de que se ha producido un error, pero si todo ha funcionado correctamente se recorren mediante un bucle del tipo *for...in* todos los contenidos que se hayan almacenado en el objeto *datos_autores*. Si ahora reproduce su película Flash, obtendrá esta información en la ventana de Salida:

```
datos_autores.onLoad=function(exito){
    if (exito){
        for (i in this){
            trace(i+": "+this[i]);
            trace("-----");
        }
    } else {
        trace("Error al cargar los datos");
    }
}

datos_autores.load("datos_autores.txt");
```

```

url2:  www.kadazuro.com
-----
mail2:  nadie@kadazuro.com
-----
pais2:  Costa Rica
-----
autor2: Carlos David Zumbado
Rodriguez
-----
url1:  www.granatta.com
-----
mail1:  mind@granatta.com
-----
pais1:  España
-----
autor1: Daniel de la Cruz Heras
-----
onLoad: [type Function]
-----

```

Como puede comprobar, la información que habíamos guardado en las variables del docu-

mento .txt se ha importado correctamente dentro de nuestra película Flash. Vaya ahora a la carpeta material/capl/fla/ contenidos_dinamicos/txt/practico/ del CD, en la que encontrará los mismos contenidos que en la carpeta anterior, pero con un documento .fla en el que vamos a dar aplicación práctica a los datos externos en vez de simplemente mostrarlos en la ventana de **Salida**. Si abre el documento carga_txt.fla, encontrará dos campos de texto con formato HTML dispuestos en la capa *textos*, así como un clip de película cuyo nombre de instancia es *cargar*, que servirá para importar los datos de datos_autores.txt. En la capa *acciones* encontrará el siguiente código:

```

datos_autores=new LoadVars();

datos_autores.onLoad=function(exito){
    if (exito){
        autor1.htmlText="<p align=\"left\"><b>"+this.autor1+"</b><br><br>"+this.pais1+"<br><a href=\"mailto:\"+this.maill+\">"+this.maill+"</a><br><a href=\"http://\"+this.url1+\"\" target=\"_blank\">http://\"+this.url1+"</a></p>";
        autor2.htmlText="<p align=\"left\"><b>"+this.autor2+"</b><br><br>"+this.pais2+"<br><a href=\"mailto:\"+this.mail2+\">"+this.mail2+"</a><br><a href=\"http://\"+this.url2+\"\" target=\"_blank\">http://\"+this.url2+"</a></p>";
    } else {
        autor1.htmlText=autor2.htmlText="Error al cargar los datos";
    }
}

cargar.onRelease=function(){
    datos_autores.load("datos_autores.txt");
}

```

Como puede observar, el código es similar al del ejemplo anterior en lo que se refiere a la carga del documento .txt (aunque en este ejemplo la carga se asocia a la pulsación del clip *cargar*), y difiere en el uso que se le da a los datos, rellenando los dos campos de texto y dándole formato para que tanto el correo como la dirección URL sean susceptibles de ser pulsados. Puede reproducir la película o abrir el documento `carga_txt.html` del mismo directorio en su navegador y pulsar el botón **cargar** para importar los datos.



Nota: La forma en que se da formato a los textos de los campos de texto *autor1* y *autor2* consiste en concatenar varias cadenas de texto, de forma que al final resulte una cadena de texto interpretable en HTML. Por ejemplo, en nuestro caso, *autor1* contiene la cadena "Daniel de la Cruz Heras", *pais1* contiene "España", *maill1* contiene "mind@granatta.com" y *url1* contiene "www.granatta.com", por lo que la siguiente cadena:

```
<p align=\"left\">
<b>"+autor1+"</b>
<br><br>"+pais1+"<br>
<a href=\"mailto:\"+
maill1+\">\"+this.maill1+\"</
a><br><a href=\"http://
\"+ url1+\"
target=\"_blank\">http:/
/\"+ url1+\"</a></p>
```

una vez que se ha sustituido el nombre de las variables por su contenido es equivalente a:

```
<p align="left">
<b>Daniel de la Cruz Heras</
b><br>
<br>
España<br>
<a href="mailto:mind@
granatta.com">
mind@granatta.com</a><br>
<a href="http://
www.granatta.com"
target="_blank">http://
www.granatta.com</a>
</p>
```

que es el texto que se interpreta en el campo de texto con formato HTML, por lo que aunque el texto que se visualiza es:

Daniel de la Cruz Heras
España
mind@granatta.com
http://www.granatta.com

tanto la dirección de correo como la dirección Web son aptas para ser pulsadas por estar escritas en formato HTML.

Carga dinámica de datos almacenados en ficheros escritos en lenguajes de marcado XML (.xml)

Por último, vamos a ver en este apartado cómo puede almacenar información externa a sus películas Flash en documentos escritos en lenguajes de marcado XML y cómo acceder a la misma utilizando el objeto XML de que dispone Flash MX. XML son las siglas de

Extensible Markup Language y se ha convertido en un estándar para el manejo de datos debido a la facilidad con la que los propios usuarios pueden crear sus *tags*, similares a los de HTML, pero que sirven para organizar la información de forma consecuente con los datos en vez de la estructura del documento. Los diferentes sistemas que pueden acceder a dicha estructura no saben nada previamente de la misma, pero si ha sido creada de forma correcta, podrá acceder a sus datos o añadirle datos nuevos sin tener que modificar los ya existentes.

Un documento XML se construye de la siguiente forma:

```
<anuncios>
<coches>
  <auto marca="Yoyota" anio="2001" />
  <auto fecha="Yuntai" anio="2002" />
</coches>
<viviendas>
  <piso calle="Margaritas"
numero="25" />
  <piso calle="Rosas Silvestres"
numero="32" />
  <piso calle="Plaza Amapolas"
numero="s/n" />
</viviendas>
</anuncios>
```

Imagine que accede a la sección de anuncios de un periódico, en donde puede encontrar coches y viviendas para alquiler o compra. Mediante una estructura XML podemos organizar jerárquicamente nuestra información en forma de árbol, por cuanto para acceder a la marca de uno de los productos anunciados en la sección antes hemos de pasar por si es un coche o una vivienda. `<anuncios>`, `<coches>` o `<viviendas>` son lo que se denominan nodos del árbol XML. Por cada uno de ellos que se abra, deberá existir otro que lo cierre, por ejemplo `</viviendas>`.

`<auto ... />` y `<piso ... />` también son nodos, pero disponen además de una serie de características que son *marca* y *anio*, en el caso de `<auto ... />`; y *calle* y *numero*, en el caso de `<piso ... />`. Esas características se denominan *atributos*, y estos nodos han de cerrarse con `" />"`.

Tenga en cuenta a la hora de crear sus documentos XML que:

```
<piso calle="Margaritas" numero="25" />
```

es lo mismo que:

```
<piso>
<calle="Margarita"/>
<numero="25"/>
</piso>
```

con la diferencia de que si en la primera expresión tenemos un nodo y dos atributos, en la segunda tenemos tres nodos. Para Flash siempre será más rápido recorrer un nodo con varios atributos en lugar de varios nodos, pero obviamente todo depende de la forma en que usted desee estructurar la información.

Para el ejemplo que vamos a ver a continuación, hemos creado un documento llamado `evolucion.xml` que puede encontrar en la carpeta `material/cap1/fla/xml/simple/` del CD. El contenido de `evolucion.xml` es el siguiente:

```
<agenda>

<noticia dia="5 de Junio de 2003">
  <datos id="1" cabecera="Carlos
Kadazuro y Dani Granatta comienzan a
escribir este libro"
url="fotografias/1.jpg" />
</noticia>
```

```

<noticia dia="14 de Agosto de 2003">
  <datos id="2" cabecera="Completada
la mitad de los capítulos del libro"
url="no" />
  <datos id="3" cabecera="Dani
Granatta regresa de las vacaciones
estivales" url="fotografias/2.jpg" />
  <datos id="4" cabecera="Carlos
Kadazuro se corta el pelo para
paliar su estrés" url="no" />
</noticia>

<noticia dia="22 de Septiembre de 2003">
  <datos id="5" cabecera="Dani
Granatta pierde el bronceado
conseguido en la playa" url="no" />
  <datos id="6" cabecera="El libro
está completamente terminado para
llevarlo a imprenta"
url="fotografias/3.jpg" />
</noticia>

<noticia dia="10 de Octubre de 2003">
  <datos id="7" cabecera="Un
productor cinematográfico hace una
oferta para adaptar el libro al
cine" url="no" />
</noticia>

</agenda>

```

Este documento .xml, que ha sido guardado con formato Unicode de igual forma a como hicimos con los documentos .txt en el apartado anterior de este capítulo, almacena información sobre diversas noticias, algunas no tan verídicas como otras :-), acerca de la realización de este libro. Cada uno de los nodos referentes a datos de cada noticia almacenan un texto llamado `cabecera`, un identificador llamado `id` y un atributo llamado `url` (estos dos últimos los utilizaremos posteriormente).

Al igual que hicimos con los contenidos almacenados en documentos .txt, vamos primeramente a importar los datos contenidos en el documento .xml en bruto, para posteriormente darle un uso más práctico. En la misma carpeta en la que está contenido `evolucion.xml`, existe un documento llamado `carga_xml fla`; ábralo y encontrará el siguiente código en el fotograma 1 de la capa *acciones*:

```

agenda_noticias=new XML();
agenda_noticias.ignoreWhite=1;

```

La primera línea nos permite crear un nuevo objeto XML llamado `agenda_noticias`, que utilizaremos para importar los datos almacenados en el documento `evolucion.xml`. Posteriormente establecemos el atributo `ignoreWhite` del objeto XML con el valor `true`, para que al acceder a la información almacenada en el fichero no se interpreten los espacios y saltos de línea entre los distintos nodos como nodos vacíos.

`parsea` es una función que explicaremos posteriormente y que se asocia al evento `onLoad` del objeto XML; es decir, las acciones de la función se ejecutan cuando se ha cargado el documento .xml (ya sea con éxito o no, al igual que hicimos con los documentos .txt).

```

parsea=function(exito){
  if (exito){
    trace(this.firstChild.nodeName);
    trace("-----");
    for (i=0;i<this.firstChild.childNodes.length;i++){
      trace(this.firstChild.childNodes[i].nodeName+" del día
"+this.firstChild.childNodes[i].attributes.dia);
      for (j=0;j<this.firstChild.childNodes[i].childNodes.length;j++){
        //trace(this.firstChild.childNodes[i].childNodes[j].nodeName);
        //nodos de nombre "datos"
        trace("-----");
        trace("id:
"+this.firstChild.childNodes[i].childNodes[j].attributes.id);
        trace("Cabecera:
"+this.firstChild.childNodes[i].childNodes[j].attributes.cabecera);
        trace("URL:
"+this.firstChild.childNodes[i].childNodes[j].attributes.url);
      }
      trace("-----");
      trace("-----");
    }
  } else {
    trace("Error al cargar los datos");
  }
}

agenda_noticias.onLoad=parsea;

```

Esta línea establece la asociación entre la función `parsea` y el evento `onLoad` del objeto `agenda_noticias`; a continuación se carga el documento `evolucion.xml` mediante una llamada al método `load` del objeto XML.

```
agenda_noticias.load("evolucion.xml");
```

Una vez que se ha cargado el fichero se ejecutan las acciones contenidas en la función `parsea`. Si el parámetro `exito` almacena el valor `false`, se muestra en la ventana de Salida un mensaje advirtiendo de que se ha producido un error en la carga de los datos contenidos en el documento `.xml`. En cambio, si `exito` almacena el valor `true`, disponemos de toda la información de `evolucion.xml` guardada en el objeto `agenda_noticias`. Veamos cómo accedemos a la misma.

El primer nodo del árbol XML, el nodo raíz, es `<agenda>`, y se referencia mediante `agenda_noticias.firstChild`, por lo que si queremos obtener el nombre del nodo, nos basta con acceder a su propiedad `nodeName`: `agenda_noticias.firstChild.nodeName`.

Cada uno de los nodos del árbol XML posee un *array* en el que se especifican los hijos de que dispone. Por ejemplo, los nodos `<coches>` y `<pisos>` eran los nodos hijos de `<anuncios>`. De igual forma, los nodos cuyo nombre es `noticia` son hijos del nodo raíz `<agenda>`, siendo `childNodes` el nombre del *array* en el que se almacena el nombre de los hijos, por lo que `agenda_noticias.firstChild.childNodes` es un *array* que se puede recorrer para ir buscando nodos

hijos de <agenda>, lo que se hace mediante un bucle `for` partiendo desde la posición 0 de `childNodes`, y mientras que la posición accedida sea inferior a la longitud del `array`.



Nota: Por ejemplo, si el `array` tiene cinco elementos, el bucle `for` recorrerá las posiciones 0, 1, 2, 3 y 4.

`agenda_noticias.firstChild.childNodes[0]` será el primer nodo hijo de <agenda> y su nombre se almacena en la propiedad `nodeName` (`agenda_noticias.firstChild.childNodes[0].nodeName` contiene "noticia"). Este nodo dispone también de un atributo llamado `dia` (`agenda_noticias.firstChild.childNodes[0].attributes.dia` contiene "5 de Junio de 2003").

Para cada nodo hijo de <agenda> crearemos otro bucle `for` para recorrer sus respectivos `arrays` de hijos, con lo que podremos acceder a los nodos de nombre `datos`. Por ejemplo, `agenda_noticias.firstChild.childNodes[0].childNodes[0]` es el primer nodo hijo del primer nodo de nombre `noticia`.

Mediante `agenda_noticias.firstChild.childNodes[0].nodeName` accedemos a su nombre (`datos`), y mediante `agenda_noticias.firstChild.childNodes[0].childNodes[0].attributes.id`, `agenda_noticias.firstChild.childNodes[0].childNodes[0].attributes.cabecera` y `agenda_noticias.firstChild.childNodes[0].childNodes[0].attributes.url`, accederemos a sus atributos `id`, `cabecera` y `url`.

Este proceso que hemos realizado en la función `parsea`, y que se conoce comúnmente como "parseo" del árbol XML, consiste en recorrer todos los nodos hijos del nodo raíz, recorriendo los nodos descendientes de cada uno de esos nodos hijos cada vez que se alcanza uno de ellos. Puede ver una interpretación visual de cómo se almacenaría la información contenida en `evolucion.xml` en la figura 1.19.

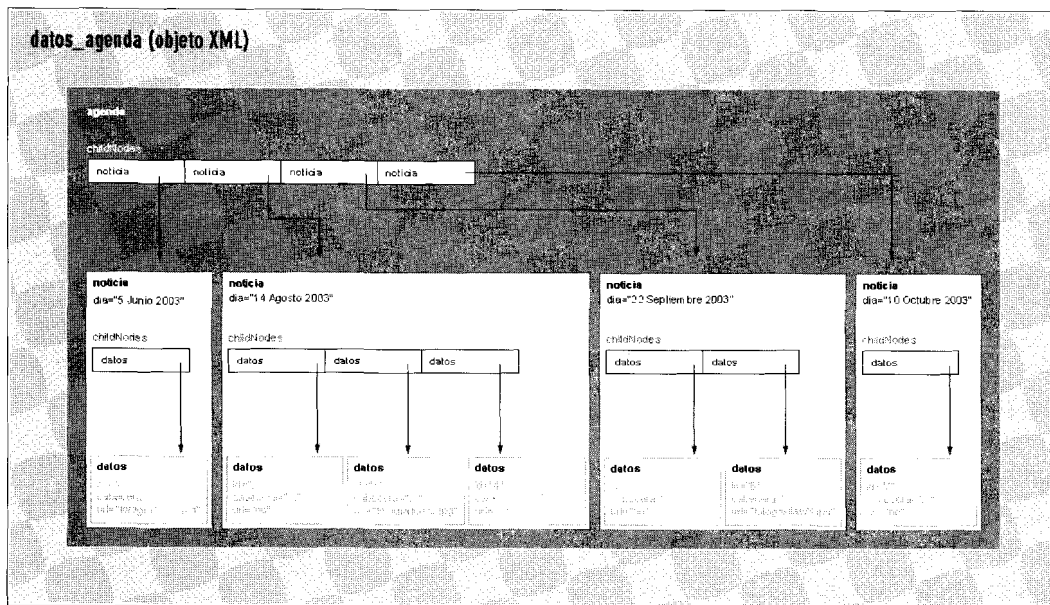


Figura 1.19.

Almacenamiento de los contenidos del documento `evolucion.xml` en el objeto XML `datos_agenda`.

Si ahora reproduce su película, obtendrá la siguiente información en la ventana de Salida:

```

agenda
-----
noticia del día 5 de Junio de 2003
-----
id: 1
Cabecera: Carlos Kadazuro y Dani Granatta comienzan a escribir este libro
URL: /fotos/1.jpg
-----
noticia del día 14 de Agosto de 2003
-----
id: 2
Cabecera: Completada la mitad de los capítulos del libro
URL: no
-----
id: 3
Cabecera: Dani Granatta regresa de las vacaciones estivales
URL: /fotos/2.jpg
  
```

```

-----
id: 4
Cabecera: Carlos Kadazuro se corta el pelo para paliar su estrés
URL: no
-----
-----
noticia del día 22 de Septiembre de 2003
-----
id: 5
Cabecera: Dani Granatta pierde el bronceado conseguido en la playa
URL: no
-----
id: 6
Cabecera: El libro está completamente terminado para llevarlo a imprenta
URL: /fotografias/3.jpg
-----
-----
noticia del día 10 de Octubre de 2003
-----
id: 7
Cabecera: Un productor cinematográfico hace una oferta para adaptar el libro al
cine
URL: no
-----
-----

```

Información que se muestra tal y como lo hemos dispuesto en la función `parsea asociada al evento onLoad del objeto XML; de ahí el que, en este caso, this.firstChild sea equivalente a agenda_noticias.firstChild.`

Ahora vamos a ver una aplicación más práctica de los datos contenidos en el documento `evolucion.xml`. Abra la carpeta `mate-`

`rial/cap1/fla/contenidos_dinamicos/xml/practico/` del CD, en la que encontrará, además de los documentos `.fla` y `.xml`, dos carpetas. La primera de ellas, `textos`, contiene siete documentos `.txt`, cada uno de los cuales almacena el texto de una noticia, según el formato que vimos en el apartado para acceder dinámicamente a contenidos guardados en documentos `.txt`. Por ejemplo, `1.txt` contiene la siguiente información:

```

&texto=En el día de hoy Carlos Kadazuro y Dani Granatta comenzaron a escribir el libro sobre Flash y contenidos dinámicos generados con PHP y MySQL, que se espera esté en las librerías para el mes de Noviembre de 2003. El libro contiene 9 capítulos y está acompañado por un CD-ROM con todos los elementos necesarios para que aprenda a conseguir la interacción entre sus películas Flash y sus bases de datos.

```

La segunda carpeta, *fotografias*, contiene tres fotografías que sirven para ilustrar tres de las siete noticias cuyo texto se encuentra en la carpeta *textos*. Lo que vamos a hacer es acceder a la información del documento *evolucion.xml* y generar un "visor" de noticias tal que, por ejemplo, el leer este nodo:

```
<datos id="1" cabecera="Carlos
Kadazuro y Dani Granatta comienzan a
escribir este libro"
url="fotografias/1.jpg" />
```

suponga el que la noticia se forme mediante el contenido del atributo *cabecera* del nodo *datos*, junto con la fotografía indicada en el atributo *url* y el texto cuyo identificador se encuentra en el atributo *id* (puesto que el valor de *id* es 1, el texto se encontrará en el documento *1.txt* de la carpeta *textos*).

Abra en su navegador el documento *carga_xml.html* de la misma carpeta y observe cómo en la parte inferior se ha creado un menú con siete opciones, una por cada uno de los nodos de nombre *datos* existentes en *evolucion.xml*. Al pulsar sobre cada uno de ellos se accede a la información textual y gráfica (si existe fotografía, puesto que algunas noticias no disponen de la misma), con lo que puede comprobar la utilidad de XML como medio de transporte de datos haciendo referencia no sólo a la información que él mismo almacena, sino también a información guardada en otros formatos.

Para ver cómo hemos creado este pequeño entorno, abra el documento *carga_xml fla*, en el que encontrará una capa llamada *borde_foto*, que se utilizará como "marco" para las fotografías y otra llamada *texto*, en la que se han dispuesto tres campos de texto con formato HTML, *nombre*, *cabecera* y *cuerpo*,

sirviendo los dos últimos para mostrar la información de cada noticia, mientras que *nombre* se usará para almacenar el nombre del nodo raíz junto con indicaciones para acceder a las noticias.

En la capa *acciones* encontrará el siguiente código:

```
news=0;
separacion=10;
anchoEscenario=450;
```

Primero creamos tres variables: *news*, que es el número de noticias que iremos encontrando en el documento *.xml*; *separacion*, que es el espacio horizontal que dejaremos entre botón y botón del menú; y *anchoEscenario*, la anchura de la película principal, y que utilizaremos posteriormente.

```
_root.createEmptyMovieClip("foto",10);
foto._x=105;
foto._y=100;
```

foto es un clip de película que se crea dinámicamente y se recoloca para posteriormente cargar en su interior las fotografías de cada noticia.

```
agenda_noticias=new XML();
agenda_noticias.ignoreWhite=1;
```

Al igual que hicimos antes, creamos un nuevo objeto XML y establecemos como *true* su atributo *ignoreWhite*, para no preocuparnos de los saltos de línea existentes en el documento *.xml*.

```
parsea=function(exito){
    if (exito){
```

Si llegamos a este punto es porque el documento *.xml* se ha cargado con éxito, por lo que ahora vamos a acceder a los datos que tiene almacenados. La idea principal es la de gene-

rar un pequeño menú, oculto hasta que esté formado, con tantos botones como noticias contenga `evolucion.xml`, así que vamos a crear este menú y, una vez creado, lo centraremos en la pantalla haciéndolo visible de nuevo.

```
_root.createEmptyMovieClip("menu",11);
```

Creamos un nuevo clip de forma dinámica al que llamamos *menu*.

```
menu._visible=0;
```

Establecemos como `false` el valor del atributo `_visible` del clip de película, con lo que éste no será visible.

```
nombre.htmlText="<p  
align=\"center\"><b>"+this.firstChild.nodeName.toUpperCase()+"</b>: Elija las  
noticias en el menú inferior de la pantalla</p>";
```

Mostramos en el campo de texto `nombre` el nombre del nodo raíz del árbol XML, así como indicaciones para ver los contenidos, para a continuación, y usando dos bucles `for` (como en el ejemplo de la carpeta `material/capl/fla/contenidos_dinamicos/xml/simple/` del CD), recorremos todo el objeto XML buscando noticias. Por cada noticia nueva que encontremos vamos a importar dentro del clip *menu* una instancia del clip `botón_noticia` que puede encontrar en la Biblioteca (figura 1.20). Si edita este clip, podrá observar cómo dispone de un campo de texto cuyo nombre de instancia es `numero`, con lo que cada vez que se importe dentro del clip *menu* se rellenará dicho campo de texto con el número de noticia correspondiente. Para importar una instancia del clip de la Biblioteca usaremos el método `attachMovie` del objeto *MovieClip*, cuya sintaxis es la siguiente:

```
nombre_del_clip.attachMovie("nombre_de_vinculación",  
"nombre_instancia",profundidad,objeto);
```

donde "nombre_instancia" es el nombre con el que se referenciará la instancia del clip de película una vez importado, profundidad es el nivel en el que se importa, "nombre_de_vinculación" es el nombre que se asocia al clip de la Biblioteca para exportarlo y usarlo mediante código, y objeto es un parámetro opcional: un objeto que puede definirse para asignarle valores a las propiedades (o crear propiedades nuevas) del clip que se crea. Para asignar un nombre de vinculación a un clip de película o botón de la Biblioteca, pulse con el botón derecho de su ratón sobre el mismo para desplegar un menú

en el que encontrará una opción de nombre Vinculación, como puede observar en la figura 1.21. Cuando elija dicha opción, se abrirá una ventana en la que podrá asignar un nombre para exportar ese clip de película; en nuestro caso, hemos asignado `nueva_noticia` como nombre de vinculación (figura 1.22).

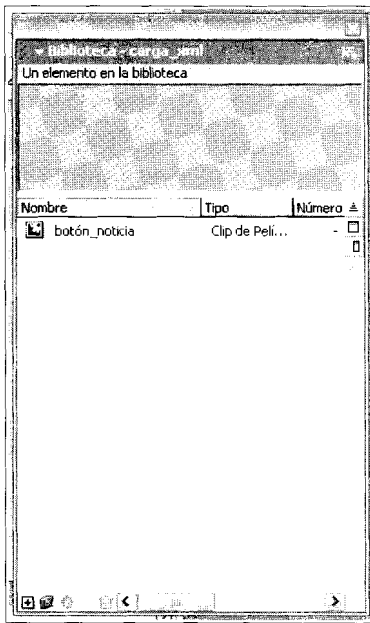


Figura 1.20.

Clip de película `botón_noticia` guardado en la Biblioteca.

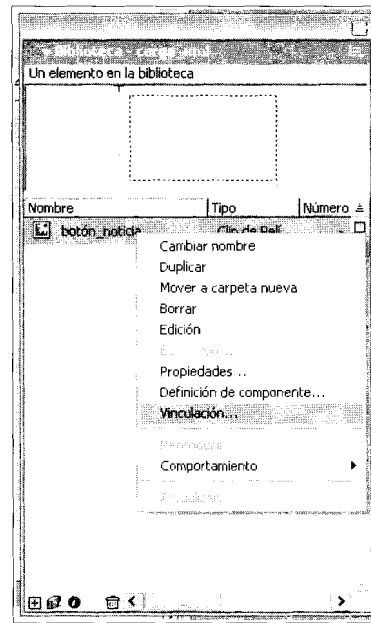


Figura 1.21.

Creación de un nombre de vinculación para el clip de película `botón_noticia`.

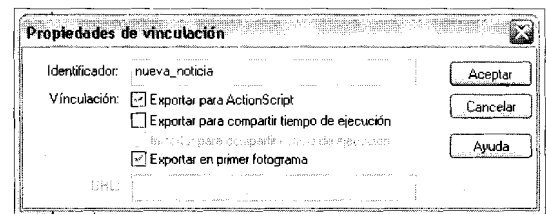


Figura 1.22.

Ventana para asignar un nombre de vinculación al clip de película `botón_noticia`.

```
for (i=0;i<this.firstChild.childNodes.length;i++){
    for (j=0;j<this.firstChild.childNodes[i].childNodes.length;j++){

        menu.attachMovie("nueva_noticia", "boton"+news,news);
    }
}
```

Por cada nodo de nombre *datos* importamos una instancia del clip `botón_noticia` de la Biblioteca (mediante su nombre de vinculación, que es `nueva_noticia`), y le asigna-

mos como nombre una cadena resultante de concatenar "boton" y el número de noticias que estamos almacenando en la variable `news`, por lo que, por ejemplo, cuando el valor

de `news` sea 0, se estará importando dentro de `menu` una instancia del clip `botón_noticia` de nombre `boton0`, para la que vamos primeramente a dotar de contenido a su campo de texto `numero`; así que insertamos en el atributo `text` del campo de texto el contenido de la variable `news` incrementado en una unidad (de forma que si `news` es 0, `numero.text` almacene 1).

```
menu["boton"+news].numero.text=news+1;
```

El modo en que hacemos referencia al clip `boton0` es el siguiente:

`menu["boton"+news]` es una expresión que varía dependiendo del valor de la variable `news`. Cuando el valor de ésta es, por ejemplo, 0, estamos haciendo referencia al clip "`boton0`" contenido dentro del clip `menu`. En realidad es exactamente igual que escribir

`menu.boton0`, pero referenciándolo de esta forma no hemos de repartir la información a los distintos clips de uno en uno después de haberlos creado, sino que puede hacerse automáticamente en el momento de importarlos desde la Biblioteca sin preocuparnos de cuántos clips habrá en total.

```
menu["boton"+news]._x=news*(menu["boton"+news]._width+separacion);
```

La línea anterior sirve para colocar las instancias del clip importadas unas al lado de otras dentro del clip `menu`, para lo que tendremos en cuenta una variable que creamos anteriormente denominada `separacion` y a la que asignamos el valor 10 y la anchura de cada una de las instancias, que es de 30 píxeles. El valor del eje horizontal en el que se colocan (`_x`) se calcula, como puede ver en la tabla 1.2, según la línea de código que precede a este párrafo.

Tabla 1.2.
Valores de colocación en el eje X de las instancias de clips importadas.

Contenido de news	Nombre de instancia asignado	Cálculos para posicionar la instancia horizontalmente	Posición en el eje horizontal (_x)
0	<code>_root.menu.boton1</code>	<code>0*(30+10)</code>	0
1	<code>_root.menu.boton2</code>	<code>1*(30+10)</code>	40
2	<code>_root.menu.boton3</code>	<code>2*(30+10)</code>	80
3	<code>_root.menu.boton4</code>	<code>3*(30+10)</code>	120
4	<code>_root.menu.boton5</code>	<code>4*(30+10)</code>	160
5	<code>_root.menu.boton6</code>	<code>5*(30+10)</code>	200
6	<code>_root.menu.boton7</code>	<code>6*(30+10)</code>	240

Con lo que las instancias de clip importadas se colocarán tal y como se muestra en la figura 1.23, en la que se puede ver cómo sería el interior del clip *menu* una vez que se haya importado el clip *botón_noticia* una vez por cada noticia (siete veces en este caso).

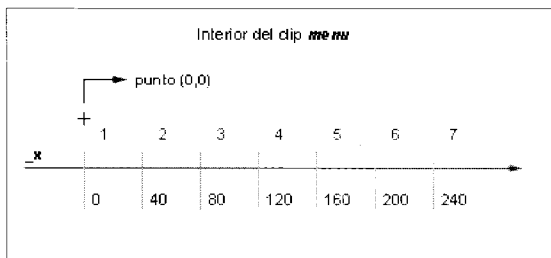


Figura 1.23.
Interior del clip menu.

Ahora vamos a asignar a cada instancia de clip importada los datos que están contenidos en los atributos de los nodos de nombre *datos*, así como el atributo *dia* contenido en los nodos de nombre *noticias* del objeto XML. La razón de asignar esos datos a variables que creamos en cada instancia es el de que cuando éstos sean pulsados se utilicen los datos contenidos en cada clip.

```
menu["boton"+news].id=this.firstChild.childNodes[i].childNodes[j].attributes.id;
menu["boton"+news].cabecera=this.firstChild.childNodes[i].childNodes[j].attributes.cabecera;
menu["boton"+news].url=this.firstChild.childNodes[i].childNodes[j].attributes.url;
menu["boton"+news].fecha=this.firstChild.childNodes[i].attributes.dia;
```

Tras haber almacenado en el clip importado los atributos *id*, *cabecera* y *url* de los nodos *datos*, y el atributo *dia* de los nodos *noticia*, vamos a declarar las acciones pertinentes para que cuando el clip sea pulsado se muestre la información correspondiente en la pantalla:

```
menu["boton"+news].onRelease=function() {
cabecera.htmlText="<p
align=\"left\"><b>"+this.fecha+".-</b>
"+this.cabecera+"</p>";
```

Lo primero que hacemos cuando pulsamos el clip es cargar el campo de texto *cabecera* con la fecha de la noticia y la cabecera que ha almacenado previamente el clip. A continuación, comprobamos el contenido de la variable *url*. Si ésta contiene la cadena "no", significa que no existe foto para la noticia, por lo que se importa la imagen *no.jpg* del directorio *fotografias* dentro del clip *foto* que creamos al comenzar. En caso contrario, se importa la imagen especificada en la variable *url*.

```

        if (this.url!="no"){
            foto.loadMovie(this.url);
        } else {
            foto.loadMovie("fotografias/no.jpg");
        }
    }

```

Después cargamos la información de la noticia para asignarla al campo de texto `cuerpo`. Dicha información se encuentra almacenada en los documentos `.txt` de la carpeta `textos`, y se accede a ellos usando el contenido de la variable `id` para cargar los datos en un objeto `LoadVars` llamado *cargador* que creamos para tal fin.

```

        cargador=new LoadVars();
        cargador.onLoad=function(exito){
            if (exito){
                cuerpo.text=cargador.texto;
            } else {
                cuerpo.text="No se pudo cargar el texto de esta
noticia";
            }
            delete cargador;
        }
        cargador.load("textos/"+this.id+".txt")
    }

```

Por último, incrementamos el valor de la variable `news` para crear nuevos clips importados correspondientes a las siguientes noticias a las que se acceda.

```

        news++;
    }
}

```

Cuando se ha creado por completo el clip `menu` con todos los clips de la Biblioteca importados y con acciones asignadas para cargar cada noticia, hemos de recolocar el clip `menu` en el centro de la pantalla, por lo que establecemos sus posiciones en `X` y en `Y` para tal fin:

```

        menu._x=(anchoEscenario/2)-(menu._width/2);
        menu._y=410;
    }

```

La forma en que se calcula la posición en X puede observarla en la figura 1.24.

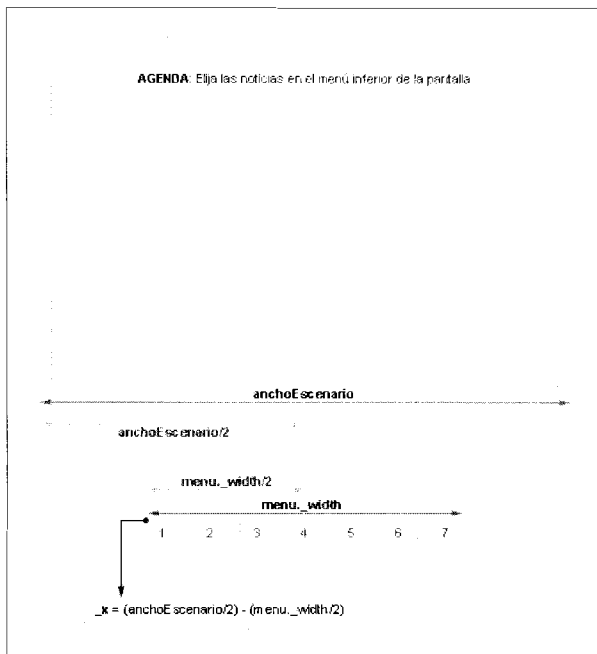


Figura 1.24.

Centrado en la pantalla del clip menu.

```
menu._visible=1;
```

Y una vez colocado el clip, se hace visible asignando el valor `true` al atributo `_visible`.

```
} else {
    nombre.htmlText="<p align=\"center\"><b>Error al cargar los datos</b></p>";
```

En caso de que no se haya cargado correctamente el documento `.xml`, se indica mediante un mensaje en el campo de texto `nombre`.

```
}
}

agenda_noticias.onLoad=parsea;
agenda_noticias.load("evolucion.xml");
```

Por último, asociamos la función `parsea` al evento `onLoad` del objeto `agenda_noticias` y cargamos el documento `evolucion.xml` para, en base a sus contenidos, comenzar a generar el entorno. Reprodúzca la película o vuelva a abrir el documento `carga_xml.html` en su navegador para observar, ahora que hemos visto cómo se ha construido, el modo en que se accede a los contenidos de cada una de las noticias (texto de cabecera almacenado en el documento `.xml`, cuerpo de la noticia en un documento `.txt` alojado en otro directorio, al igual que imagen de la noticia, en formato de imagen `.jpg`).

Si deseara incluir documentos de otro tipo de los analizados en este capítulo a nuestra pequeña agenda de noticias, podría añadir un atributo a cada nodo en el que, por ejemplo, hiciera referencia a la ruta de distintos documentos MP3 que se importasen dinámicamente a la película de igual forma a como lo hacen las fotografías.

Explicadas las nociones básicas necesarias para importar contenidos a Flash de forma dinámica, vamos a dedicar los siguientes capítulos del libro a relatar el modo en que puede crear sus propias bases de datos y

cómo acceder a los contenidos que se encuentran almacenados en las mismas.

```
gotoAndPlay("capitulo_2");
```

Capítulo 2

Instalación de servidores

Además de una licencia o versión de prueba de treinta días de Flash MX (que puede encontrar en la carpeta `/software Macromedia` del CD que acompaña a esta publicación), para completar los ejercicios y prácticas de este libro necesitará disponer de los siguientes elementos instalados en su ordenador:

- Servidor Apache.
- PHP.
- Servidor de bases de datos MySQL.
- Administrador de bases de datos phpMyAdmin.

Vamos a instalar todo este software en nuestra máquina para poder trabajar de forma local, con lo que no necesitaremos subir mediante FTP a un servidor ubicado en Internet los documentos que cree para comprobar si estos

funcionan, sino que las llamadas serán referidas al servidor local (*localhost*) del que dispondremos una vez terminada la instalación.

Hemos dividido el capítulo en dos apartados, el primero de ellos consiste en la instalación en ordenadores con sistema operativo Windows, y el segundo en ordenadores Macintosh.

Instalación de servidores en ordenadores con Windows

El software que utilizaremos si disponemos de un sistema operativo Windows puede instalarse de dos formas. La primera de ellas consiste en instalar cada programa por separado y modificar los archivos de configuración de cada uno para ponerlos en funcionamiento. La segunda consiste en utilizar "instaladores" que automaticen el proceso de instalación

conjunta de todas las aplicaciones que necesitamos. Uno de los mejores y más conocidos, y cuya puesta en marcha detallaremos en este apartado, es AppServ, un proyecto a cuyo responsable desde Octubre del 2001, Phanupong Panyadee (apples), agradecemos enormemente su autorización para incluir la última versión (AppServ 2.1.0, en el momento de escribir estas líneas) de la aplicación en el CD que acompaña a este libro.

Si desea descargar una versión posterior del proyecto, o simplemente obtener información acerca de mejoras efectuadas sobre el mismo, puede acceder a su sitio Web en Internet <http://www.appservnetwork.com>.

Vamos a comenzar la instalación de nuestra versión de AppServ, que incluye el siguiente software:

- PHP 4.3.3RC1.
- MySQL 4.0.13.
- PhpMyAdmin 2.5.1.

Vaya a la carpeta /material/instaladores/ del CD, en la que encontrará un fichero de nombre `appserv-win32-2.1.0.exe`. Haga doble clic sobre el mismo para acceder a la pantalla de bienvenida de la instalación de AppServ (figura 2.1).

Pulse el botón **Siguiente (Next)** para acceder a otra pantalla en la que se le solicitará que introduzca la ruta de su disco duro en la que desea instalar los servidores (figura 2.2). Nosotros vamos a elegir la ruta predeterminada (`c:\appserv`), pero puede teclear la que usted desee; eso sí, recuerde la dirección que seleccione, puesto que de ahora en adelante deberá colocar en la carpeta `www` del directorio de instalación que elija los archivos de las distintas prácticas que realicemos.

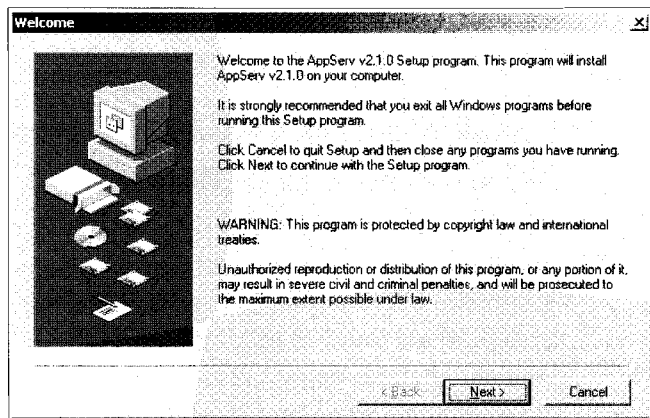


Figura 2.1.

Pantalla de bienvenida de la instalación de AppServ.

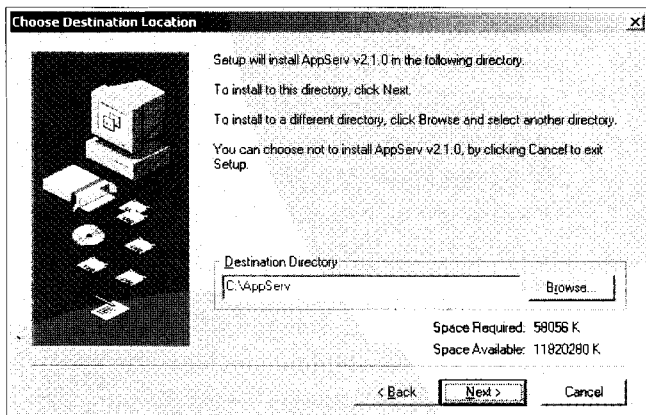


Figura 2.2.

Selección de carpeta de instalación.

Después de pulsar el botón **Siguiente** habrá de elegir el tipo de instalación que desea realizar (figura 2.3). Elija la instalación Personalizada (Custom) para observar qué material hay disponible e instalar sólo aquel que vaya a necesitar. Posteriormente, pulse el botón **Siguiente**.

Seleccione las opciones Apache, PHP, MySQL y phpAdmin, y pulse **Siguiente** para continuar (figura 2.4).

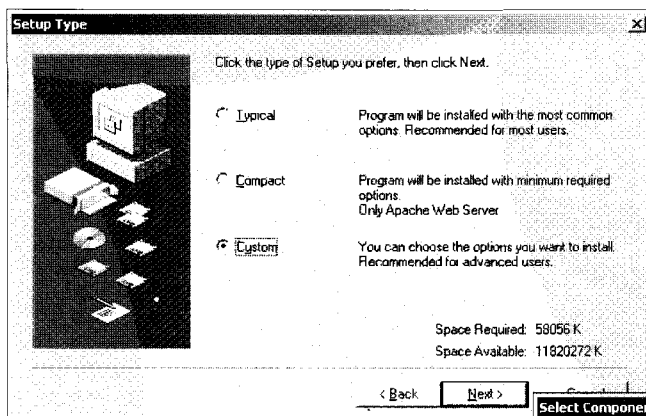
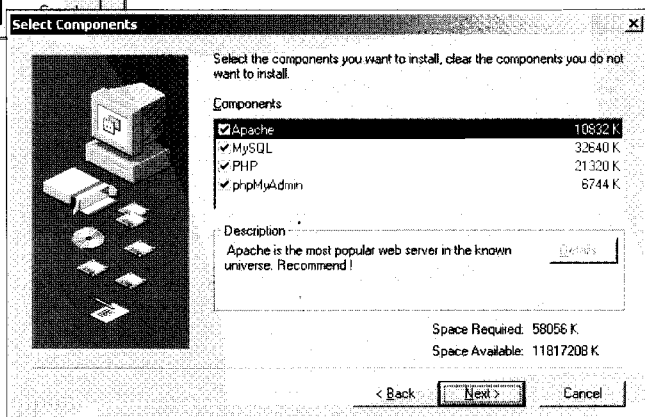


Figura 2.3.

Selección del tipo de instalación.

Figura 2.4.

Selección de los programas a instalar en el ordenador.



La siguiente pantalla (figura 2.5) sirve para establecer la configuración del servidor Apache. Introduzca `localhost` como nombre del servidor y `admin@localhost` como correo del administrador. Si no va utilizar el servicio de correo, la dirección con la que rellene el campo de correo del administrador no tiene mucha importancia. Teclee 80 para el número de puerto, aunque si ya tiene otro servidor Web instalado previamente puede teclear 81 para especificar el uso del puerto 81, con la salvedad de que habrá de hacerle referencia en las llamadas al servidor:

- `http://localhost` (si el servidor Web está configurado en el puerto 80).
- `http://localhost:81` (si el servidor Web está configurado en el puerto 81).

Una vez configurado el servidor Apache pulse en **Siguiente** para llegar a la pantalla de configuración del servidor de bases de

datos MySQL (figura 2.6), en la que especificaremos tanto el nombre de usuario como la contraseña. Nosotros rellenaremos con la palabra "root" tanto el campo **usuario** como el campo **password**, por ser local el uso de este servidor. Si su servidor va a ser de acceso público le recomendamos que utilice nombres de usuario y contraseñas que sean complicados de averiguar. La opción **Charset** le servirá para seleccionar la configuración de caracteres que desee (puede obtener información sobre las configuraciones disponibles en la dirección Web <http://www.w3.org/International/O-charset-lang.html>); nosotros elegiremos la que se especifica por defecto, "latin1", correspondiente a la mayoría de lenguajes de Europa Occidental (castellano, inglés, francés, etc.).

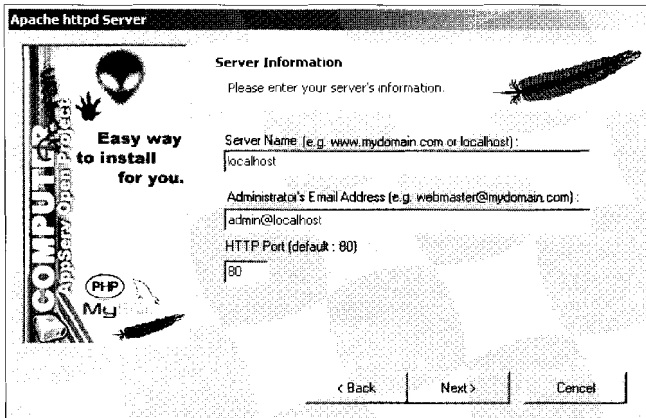


Figura 2.5.

Configuración del servidor Web Apache.

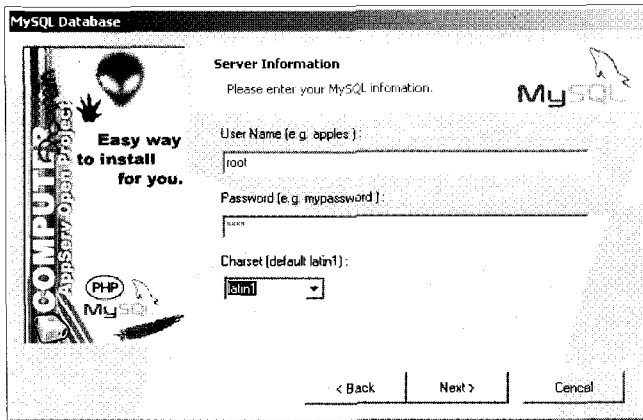


Figura 2.6.

Pantalla de configuración del servidor MySQL.

Pulse el botón **Siguiente** y espere hasta que se hayan copiado completamente todos los ficheros en su disco duro (figura 2.7).

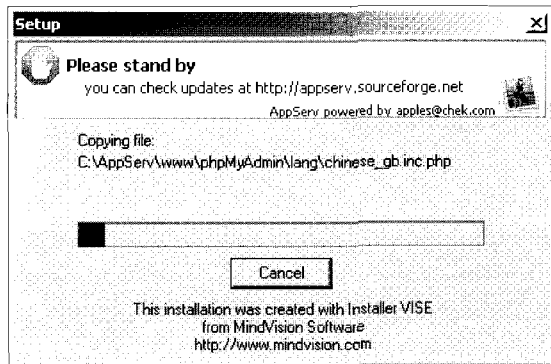


Figura 2.7.

AppServ copiando los ficheros al ordenador.

Cuando AppServ haya terminado de copiar los ficheros en su ordenador será preguntado sobre si desea iniciar el funcionamiento de los servidores Apache y MySQL (figura 2.8). Si lo desea, asegúrese de seleccionar ambas opciones.

Si desea iniciar el funcionamiento de MySQL cada.cada vez que encienda su ordenador, seleccione el icono en forma de semáforo que aparece en el Área de notificación de Windows (en la esquina inferior derecha, junto al reloj que marca la hora del sistema y demás iconos pertenecientes a programas que tenga instalados) para abrir la consola de administración de MySQL (WinMySQLAdmin). Una vez que se encuentre dentro de la misma, presione el

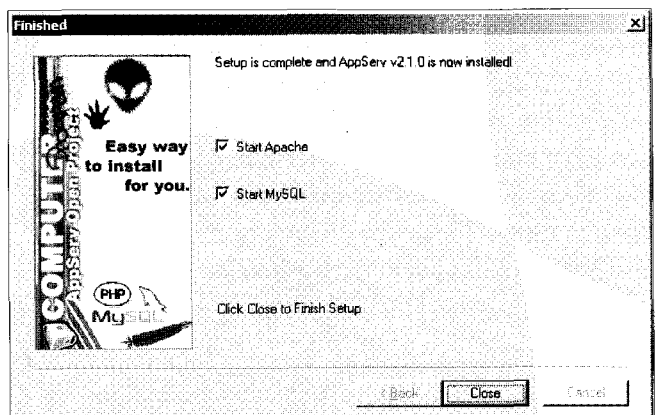


Figura 2.8.

Fin de la instalación.

botón **Create ShortCut on Start Menu** (crear un acceso directo en el menú Inicio) que se encuentra en la pestaña **my.ini Setup** del administrador (figura 2.9).

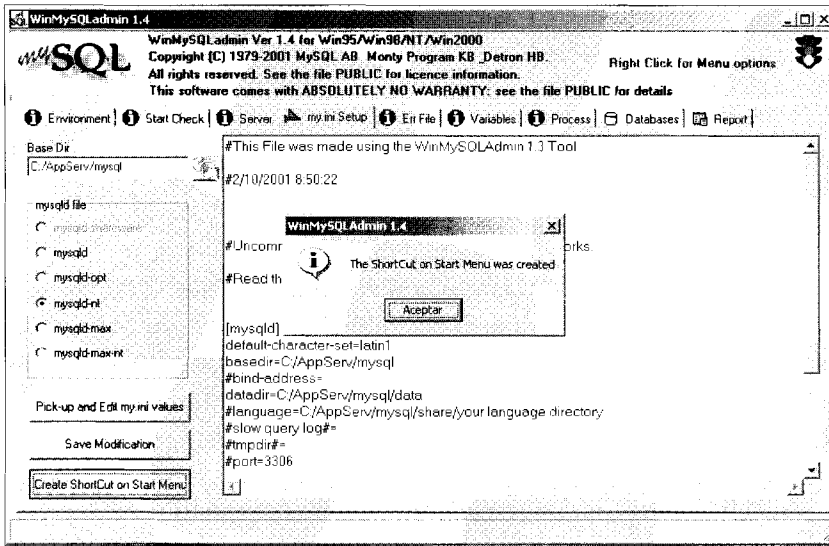


Figura 2.9.
Configuración de MySQL para que se inicie al comenzar Windows.

Una vez concluida la instalación, puede comprobar si su servidor está funcionando correctamente tecleando `http://localhost` en su navegador, en el que deben aparecer contenidos similares a los de la figura 2.10.

Por último, configuraremos phpMyAdmin para que funcione con nuestro servidor y nuestras bases de datos. Edite el documento `config.inc.php` que se encuentra en la carpeta `www/phpMyAdmin` del directorio en el que haya realizado la instalación de AppServ (por ejemplo, en nuestro caso se trata del documento `c:/appserv/www/phpMyAdmin/config.inc.php`) y busque la siguiente línea:

```
$cfg['PmaAbsoluteUri'] = '';
```

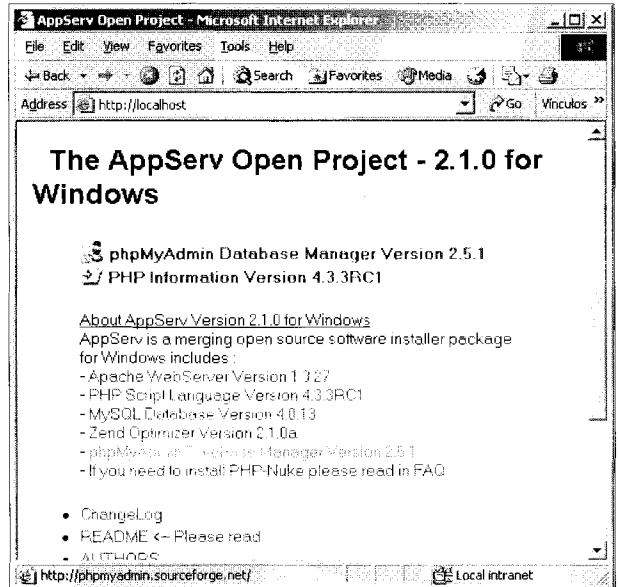


Figura 2.10.
Prueba de funcionamiento del servidor.

Éste es el lugar en el que establecer la ruta absoluta en el servidor Web hasta el directorio que contiene phpMyAdmin; por tanto, sustitúyala por la línea siguiente:

```
$cfg['PmaAbsoluteUri'] = 'http://localhost/phpMyAdmin/';
```

Busque ahora la línea:

```
$cfg['Servers'][$i]['password'] = '';
```

Y sustitúyala por ésta, en la que establece que la contraseña para el manejo de la base de datos es la que introdujo al instalar AppServ ("root"):

```
$cfg['Servers'][$i]['password'] = 'root';
```

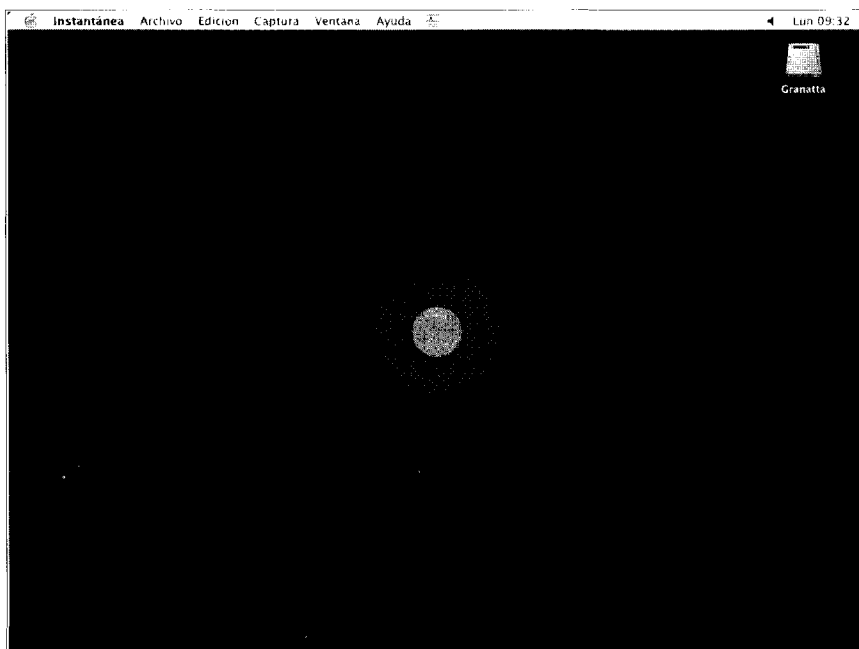
Tras salvar el fichero tecleamos ahora en nuestro navegador la siguiente dirección <http://localhost/phpMyAdmin>, con lo que accederemos con éxito al administrador phpMyAdmin.

Instalación de servidores en ordenadores Macintosh

La versión 10.2 (Jaguar) del sistema operativo OS X de Macintosh (figura 2.11), pese a que el sistema mantenga el aspecto gráfico habitual de las máquinas de Apple, dispone de un núcleo basado en Unix, por lo que permite el uso de algunas herramientas estándar de dicho sistema operativo, como puedan ser Apache, PHP y MySQL. Cuando instale Mac OS X 10.2, se instalará por defecto el software correspondiente a un servidor Web Apache, prácticamente listo para ser utilizado excepto por unas pequeñas modificaciones que debemos realizar antes de comenzar. Supongamos que disponemos de una cuenta en nuestro ordenador Macintosh cuyo nombre es *granatta* y cuyo password es "Granatta", no olvide sustituir ambos por los suyos propios cuando esté trabajando con su máquina:

Figura 2.11.

Imagen del Escritorio en Mac OS X 10.2.



Activación del módulo de PHP

Lo primero que haremos será activar el módulo de PHP que se instala con Apache para posteriormente comprobar que nuestra instalación ha sido exitosa y realizar pruebas con nuestro ordenador funcionando como un servidor en el que se interpreten los documentos .php que creemos. Para ello, vaya a la

carpeta Aplicaciones/Utilidades de su sistema (figura 2.12) y ejecute la aplicación Terminal (figura 2.13). Como puede observar, en la ventana de su terminal dispone de un *shell* de UNIX para ejecutar comandos.

Empezaremos por modificar el archivo de configuración de Apache tecleando lo siguiente:

```
sudo pico /etc/httpd/httpd.conf
```

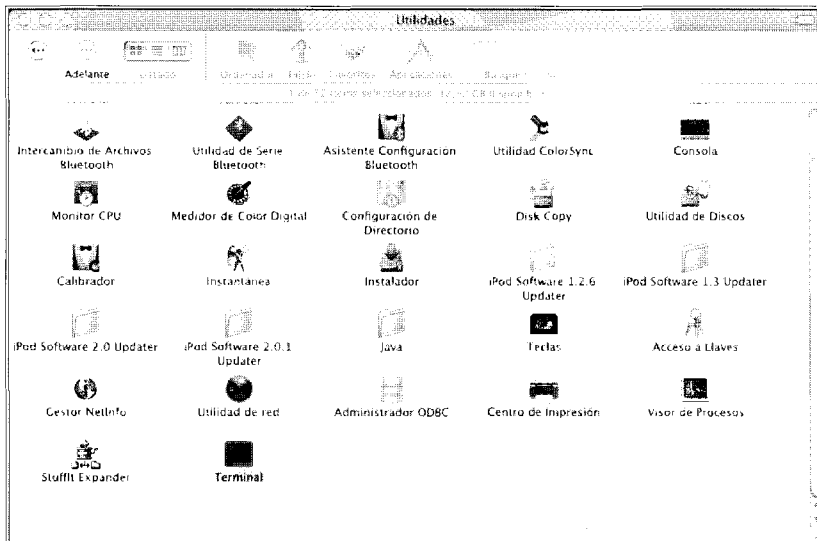


Figura 2.12.
Selección de la aplicación Terminal.

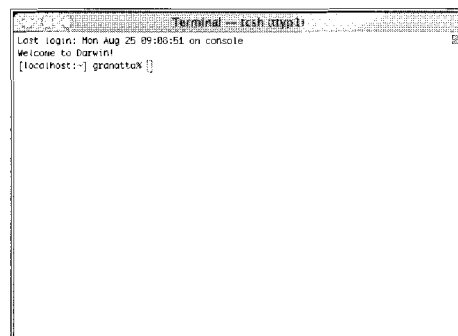


Figura 2.13.
Shell de UNIX para ejecutar comandos.

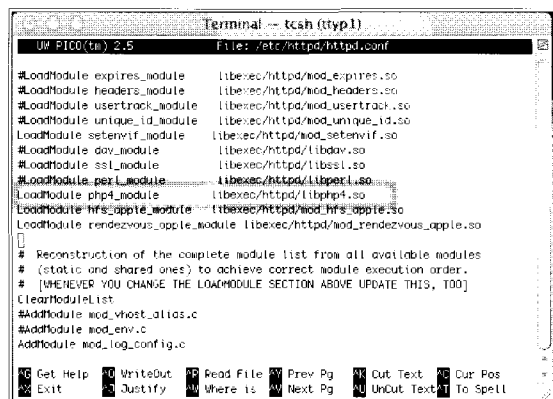


Nota: sudo es un comando que permite la llamada a otros comandos como superusuario, mientras que pico es el nombre del editor de texto que vamos a utilizar para modificar el fichero de configuración.

Teclee la contraseña de su cuenta de usuario en Macintosh cuando le sea solicitada y, una vez dentro del documento, desplácese con las flechas de cursor hacia la parte inferior hasta que encuentre las siguientes dos líneas:

```
LoadModule php4_module libexec/  
httpd/libphp4.so  
AddModule mod_php4.c
```

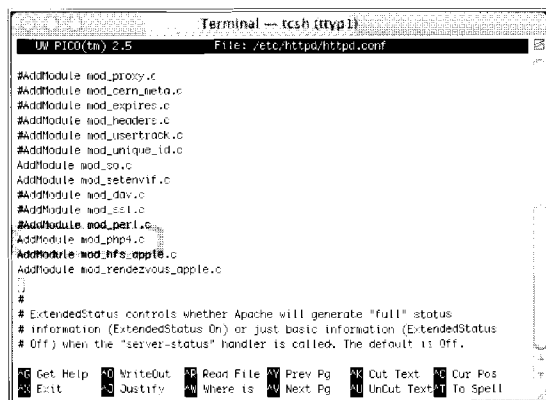
para eliminar el carácter "#", que sirve para que sean consideradas líneas de comentario (figuras 2.14 y 2.15).



```
Terminal -- tcsh (tty1)  
UP PICO(tm) 2.5 File: /etc/httpd/httpd.conf  
#LoadModule expires_module libexec/httpd/mod_expires.so  
#LoadModule headers_module libexec/httpd/mod_headers.so  
#LoadModule usertrack_module libexec/httpd/mod_usertrack.so  
#LoadModule unique_id_module libexec/httpd/mod_unique_id.so  
LoadModule setenvif_module libexec/httpd/mod_setenvif.so  
#LoadModule dav_module libexec/httpd/libdav.so  
#LoadModule ssl_module libexec/httpd/libssl.so  
#LoadModule perl_module libexec/httpd/libperl.so  
LoadModule php4_module libexec/httpd/libphp4.so  
LoadModule hfs_apple_module libexec/httpd/mod_hfs_apple.so  
LoadModule rendezvous_apple_module libexec/httpd/mod_rendezvous_apple.so  
#  
# Reconstruction of the complete module list from all available modules  
# (static and shared ones) to achieve correct module execution order.  
# [WHENEVER YOU CHANGE THE LOADMODULE SECTION ABOVE UPDATE THIS, TOO]  
ClearModuleList  
AddModule mod_vhost_alias.c  
AddModule mod_env.c  
AddModule mod_log_config.c
```

Figura 2.14.

Modificación del fichero de configuración de Apache (I).



```
Terminal -- tcsh (tty1)  
UP PICO(tm) 2.5 File: /etc/httpd/httpd.conf  
#AddModule mod_proxy.c  
#AddModule mod_cern_meta.c  
#AddModule mod_expires.c  
#AddModule mod_headers.c  
#AddModule mod_usertrack.c  
#AddModule mod_unique_id.c  
AddModule mod_so.c  
AddModule mod_setenvif.c  
#AddModule mod_dav.c  
#AddModule mod_ssl.c  
#AddModule mod_perl.c  
AddModule mod_php4.c  
AddModule mod_hfs_apple.c  
AddModule mod_rendezvous_apple.c  
#  
# ExtendedStatus controls whether Apache will generate "full" status  
# information (ExtendedStatus On) or just basic information (ExtendedStatus  
# Off) when the "server-status" handler is called. The default is Off.  
#  
# Get Help WriteOut Read File Prev Pg Out Text Cur Pos  
# Exit Justify Where is Next Pg UnOut Text To Spell
```

Figura 2.15.

Modificación del fichero de configuración de Apache (II).

Bajo la línea cuyo texto es:

```
AddType application/x-tar.tgz ()
```

añada las dos líneas siguientes:

```
AddType application/x-httpd-php .php  
AddType application/x-httpd-php-  
source .phps
```

Busque ahora las líneas cuyo texto es:

```
<IfModule mod_dir.c>  
DirectoryIndex index.html  
</IfModule >
```

cuya utilidad es la de establecer los documentos que el servidor Web reconocerá como la primera página a cargar del dominio (figura 2.16) y añada los nombres de documento index.php e index.htm de forma que el contenido de la línea sea:

```
<IfModule mod_dir.c>  
DirectoryIndex index.html index.php  
index.htm  
</IfModule >
```

Presione conjuntamente las teclas **Control-X** para guardar el documento. Cuando el terminal le pregunte si desea guardar los cambios, presione la tecla **Y**, y posteriormente pulse **Intro** para confirmar que el nombre del documento que se modifica es `httpd.conf`. Posteriormente, volverá usted al *shell* de UNIX de su terminal.

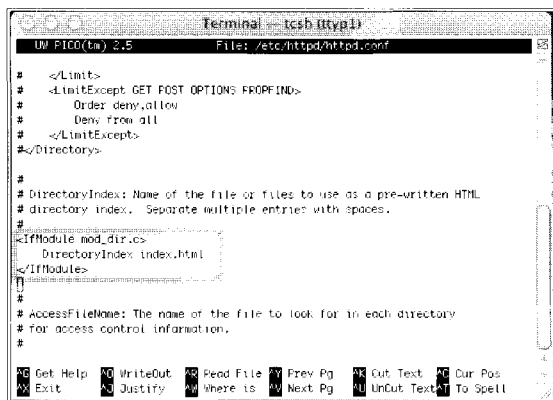


Figura 2.16.

Modificación del fichero de configuración de Apache (y III).

Iniciar y detener el servidor Web Apache

Una vez que ha activado el módulo de PHP de su sistema puede iniciar el funcionamiento de su servidor Web Apache mediante la opción **Compartir** de su carpeta de **Preferencias del Sistema** (figura 2.17).

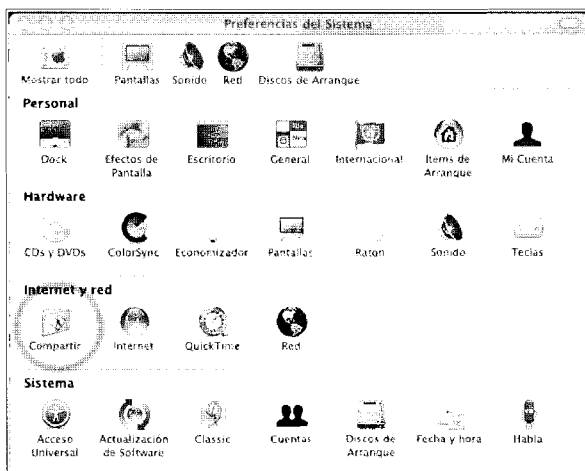


Figura 2.17.

Carpeta de Preferencias del Sistema.

Una vez que ha elegido dicha opción, diríjase a la pestaña de nombre **Servicios** (figura 2.18) y marque la casilla cuyo texto reza **Compartir Web** (figura 2.19).

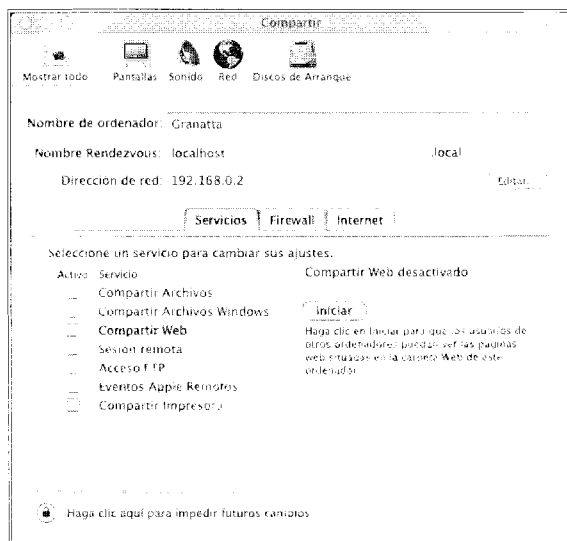


Figura 2.18.

Puesta en funcionamiento del servidor Apache (I).

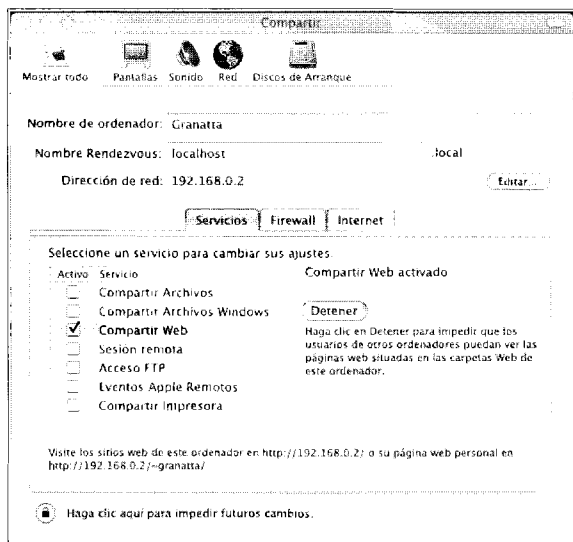


Figura 2.19.

Puesta en funcionamiento del servidor Apache (y II).



Nota: Los documentos que en este ejemplo se visualizarán en el host `http://localhost/~granatta` se encuentran en la carpeta `Users/granatta/Web` de su sistema.

En ese momento se le asignará una dirección IP que le permitirá visualizar el contenido de los documentos de su servidor, tal y como se muestra en la figura 2.20, pudiendo detener el funcionamiento del servidor pulsando sobre el botón de texto **Detener**.

Ahora comprobaremos si su servidor Apache está funcionando correctamente, es decir, si es capaz de interpretar documentos .php, para lo que vamos a utilizar el editor de texto TextEdit (disponible en la carpeta Aplicaciones de su sistema).

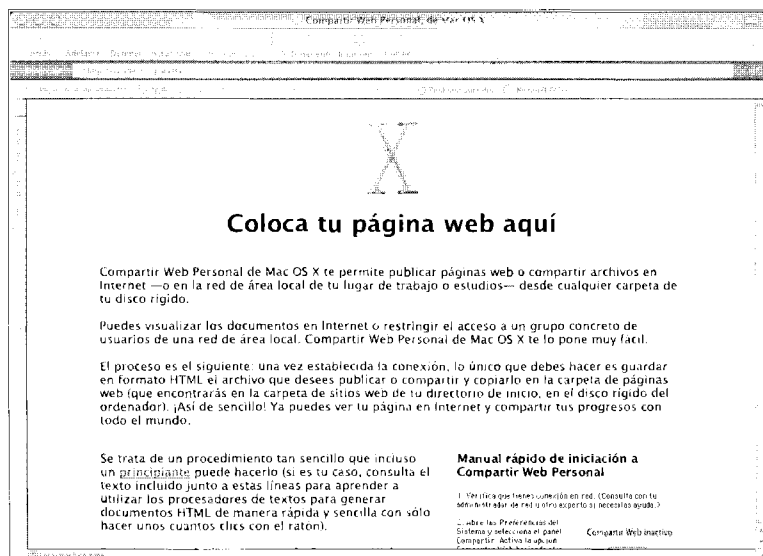


Figura 2.20.

Documento Web en la raíz del servidor.



Nota: `phpinfo()` es una función predeterminada del lenguaje PHP que le permite obtener información acerca del sistema sobre el que está corriendo el documento que realiza la llamada a la función.

Cree un nuevo documento de texto y teclee lo siguiente (figura 2.21):

```
<?
phpinfo();
?>
```

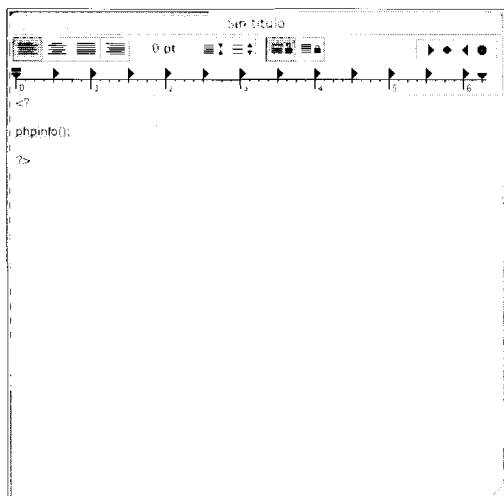


Figura 2.21.

Prueba de funcionamiento del servidor Web.

Por defecto, la aplicación TextEdit guarda los documentos con formato de texto enriquecido (con extensión .rtf), así que vamos a convertir lo que acabamos de escribir a texto plano (Menú Formato>Convertir texto normal) y posteriormente guardaremos el fichero (Menú Archivo>Guardar como) con el nombre de `info.php` en la carpeta `Users/granatta/`

Web (figura 2.22), que es el directorio raíz de nuestro espacio personal en el servidor Web.

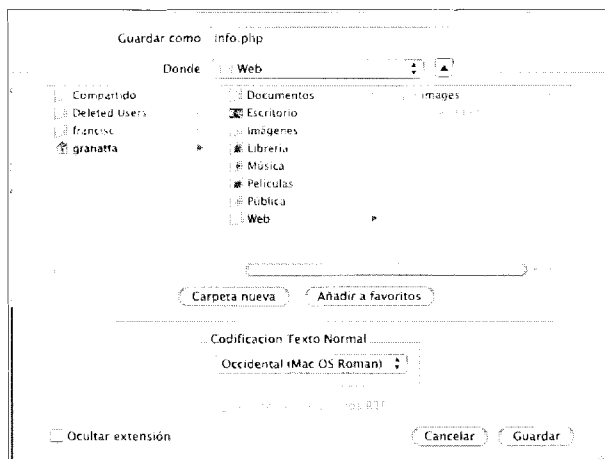


Figura 2.22.

Opciones para guardar el documento de prueba del servidor Web.

Cuando sea preguntado acerca de si quiere añadir la extensión .txt al documento que acaba de salvar, elija la opción **No añadir** (figura 2.23), con lo que si teclea en su navegador la dirección `http://localhost/~granatta/info.php` debe ver en su pantalla unos contenidos similares a los de la figura 2.24, información relativa a la versión de PHP que acaba de activar y probar con éxito.

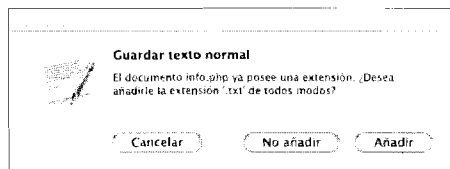


Figura 2.23.

Cuadro de diálogo al guardar el documento de prueba.

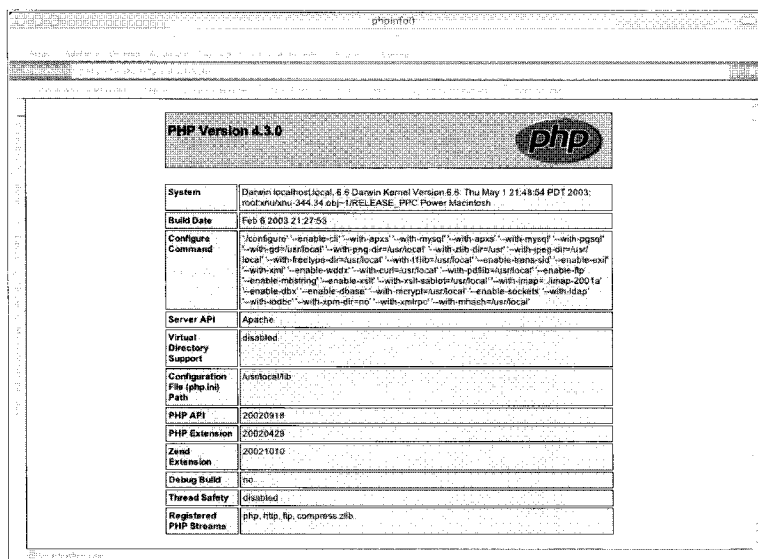


Figura 2.24.
Información sobre la versión de PHP instalada en el servidor.

Creación de una cuenta MySQL

Antes de instalar MySQL debe crear una cuenta y un grupo, ambos con el nombre *mysql* (bajo la que correrán las bases de datos MySQL), aunque por defecto dicha cuenta y grupo son creadas por la instalación de OS X 10.2. Para verificar si dispone de la cuenta y grupo

mysql, abra la aplicación Gestor NetInfo (presente en la carpeta Aplicaciones/Utilidades de su sistema, como puede ver en la figura 2.25). Una vez dentro, encontrará en su pantalla un panel dividido en tres columnas.

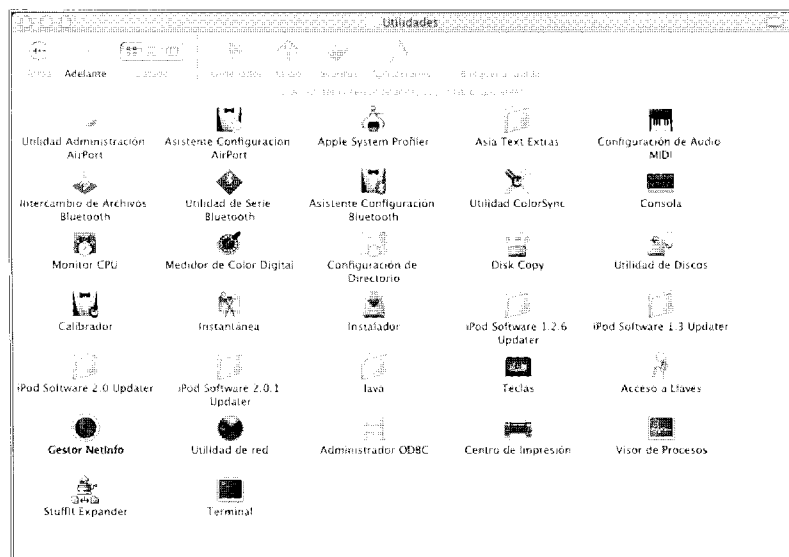


Figura 2.25.
*Selección de la aplicación
Gestor NetInfo.*

Seleccione **users** en la segunda para comprobar cómo en la tercera columna existe un usuario de nombre *mysql* (figura 2.26), y haga lo mismo con **groups** para verificar que existe un grupo llamado *mysql* en la tercera columna (figura 2.27).

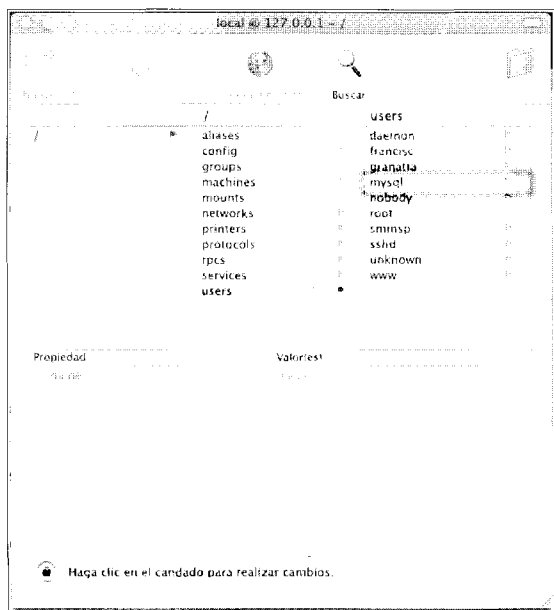


Figura 2.26.

Usuarios existentes en el sistema.

En caso de que no disponga de un usuario ni un grupo de nombre *mysql*, cierre la aplicación Gestor NetInfo y abra la opción **Cuentas** de las **Preferencias del Sistema**. Elija **Nuevo usuario**, teclee **MySQL Server** en **Nombre** y **mysql** en **Nombre corto**, no añada contraseña ni marque **Permitir al usuario administrar este ordenador** o **Permitir al usuario iniciar sesión desde Windows**. Pulse el botón **Ok** y, cuando el sistema le advierta de que **Este usuario no ha definido ninguna contraseña**, pulse **Ignorar**.

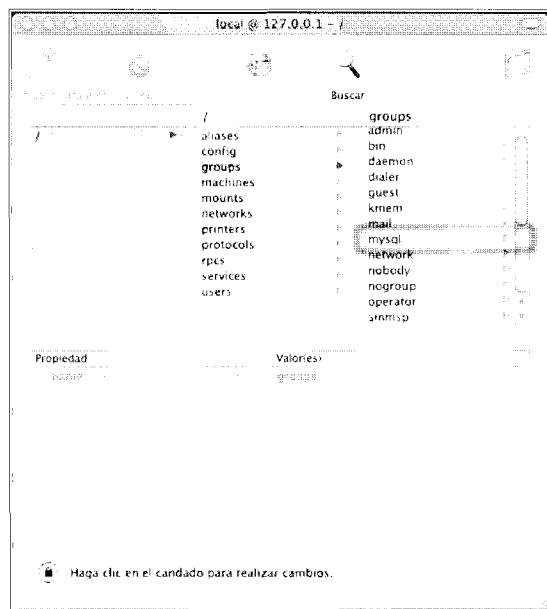


Figura 2.27.

Grupos existentes en el sistema.

Abra de nuevo la aplicación **Gestor NetInfo** y autentifíquese (**Menú Seguridad > Autenticar**, figura 2.28) mediante su contraseña de usuario (en nuestro caso, "*Granatta*").

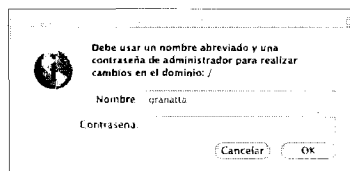


Figura 2.28.

Autenticación de usuario para realizar cambios.

Vaya a la segunda columna y pulse sobre `users`, para buscar al usuario de nombre `mysql` en la tercera. Una vez seleccionado, debe ver una pantalla similar a la de la figura 2.29, en la que se muestran las propiedades de la cuenta `mysql`, sobre cuyas entradas podrá efectuar cambios una vez que se haya autenticado como usuario.

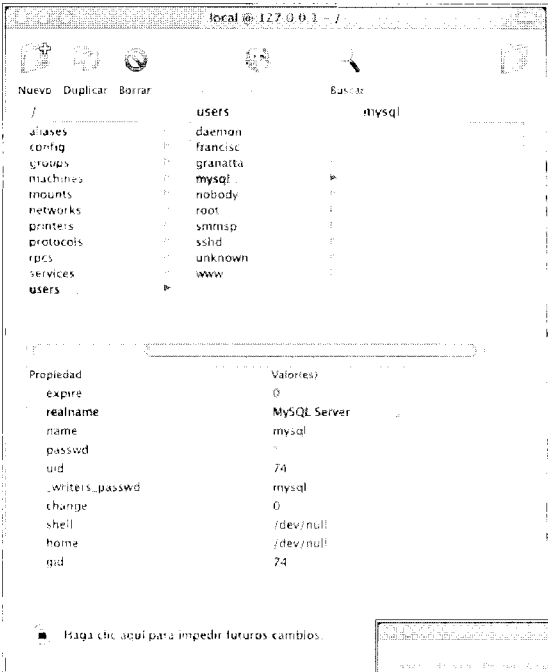


Figura 2.29.
Propiedades de la cuenta de usuario mysql.

Instalación de la base de datos MySQL

Para instalar la base de datos MySQL vamos a echar mano de la ayuda que encontraremos en el magnífico sitio Web de Marc Liyanage, cuya dirección Web es <http://www.entropy.ch/software/macosex> (figura 2.30).

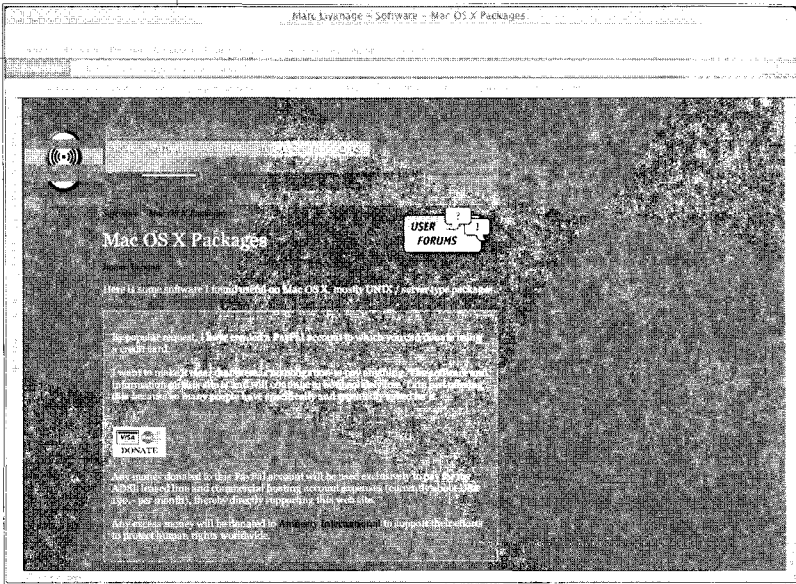


Figura 2.30.
Sitio Web de Marc Liyanage.

Una vez haya accedido a la URL que hemos indicado en el párrafo anterior, pulse sobre la opción MySQL Database Server, accediendo al documento <http://www.entropy.ch/software/macosex/mysql/> (figura 2.31), en el que encontrará las instrucciones necesarias para descargar e instalar la base de datos MySQL en su ordenador.

En primer lugar, descargue desde el sitio Web oficial de MySQL el documento binario precompilado de MySQL en <http://www.mysql.com/downloads/mysql-4.0.html>. Elija Standard en la sección Mac OS X Package Installer Downloads (asegúrese de que es Package Installer Downloads y no Downloads, figura 2.32).

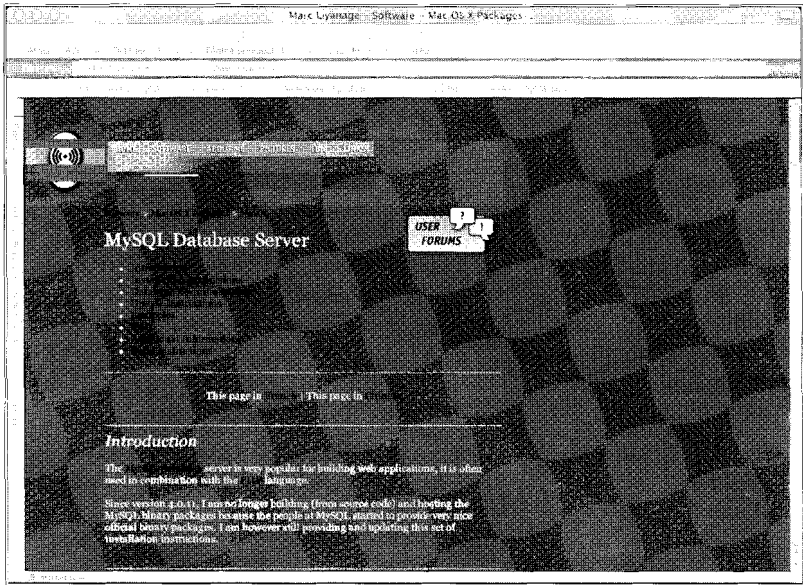
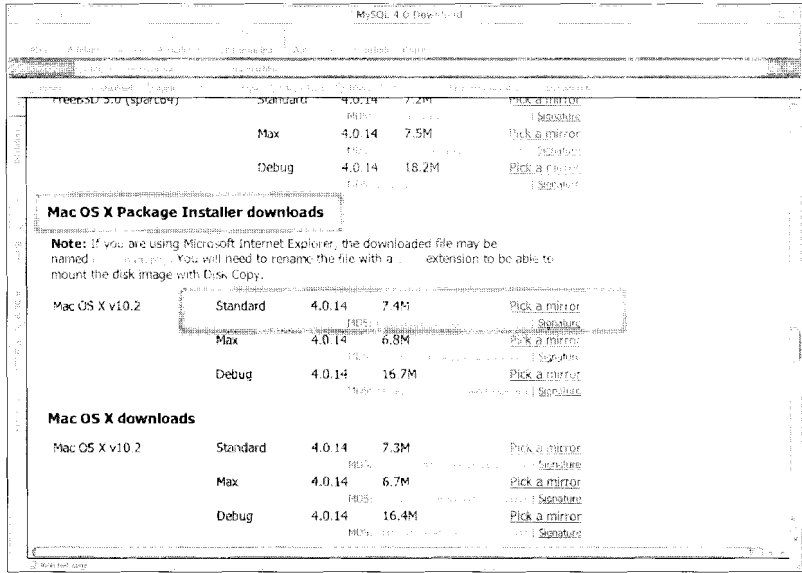


Figura 2.31.
Instrucciones para descargar e instalar la base de datos MySQL.

Figura 2.32.
Descarga de documento binario precompilado de MySQL.



Una vez descargado el documento (por ejemplo, en este caso hemos descargado la versión 4.0.13), haga doble clic sobre el mismo para montarlo (el documento descargado, que puede ver en la figura 2.33, es una imagen de disco que debe ser montada, lo que será realizado automáticamente por la aplicación Disk Utility de Mac OS X cuando haga doble clic sobre el documento .dmg).

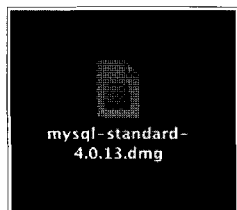


Figura 2.33.

Documento binario .dmg descargado al disco.

Abra la imagen del disco (figura 2.34) y haga doble clic sobre el instalador que encontrará dentro (mysql-standard-4.0.13.pkg, figura 2.35), que instalará todos los ficheros necesarios en el directorio /usr/local/mysql-4.0.13 después de haberle pedido su contraseña (figura 2.36) y haber seguido uno por uno los pasos de instalación que se le indiquen (figuras 2.37, 2.38 y 2.39).

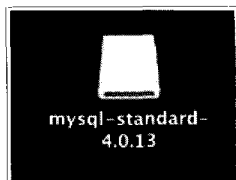


Figura 2.34.

Imagen del disco.

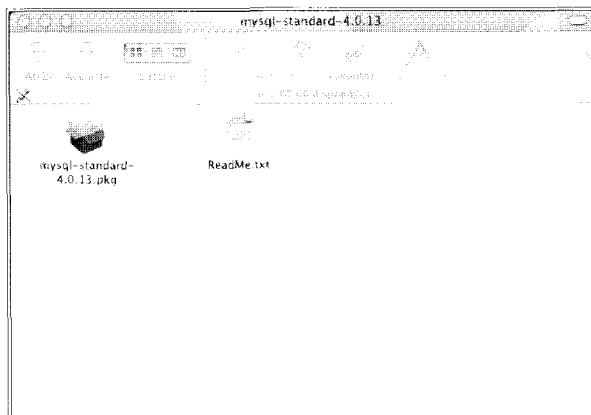


Figura 2.35.

Instalador de la base de datos MySQL.

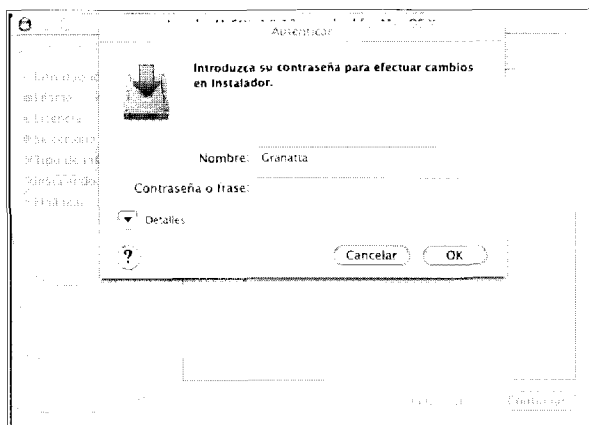


Figura 2.36.

Petición de contraseña para instalar la base de datos MySQL.

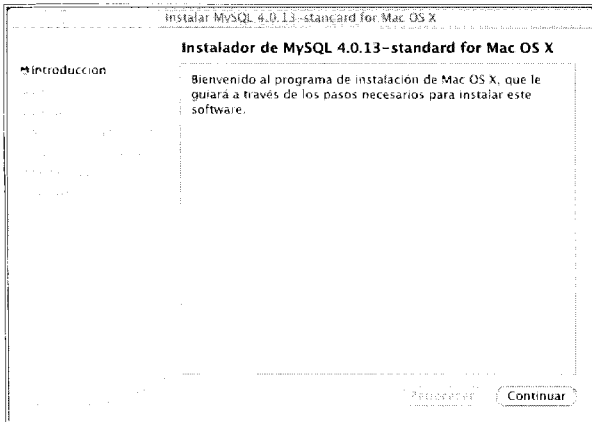


Figura 2.37.

Instalación de la base de datos MySQL (I).

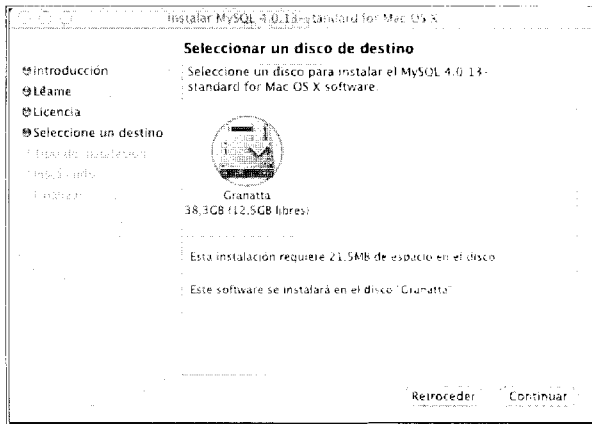


Figura 2.38.

Instalación de la base de datos MySQL (II).

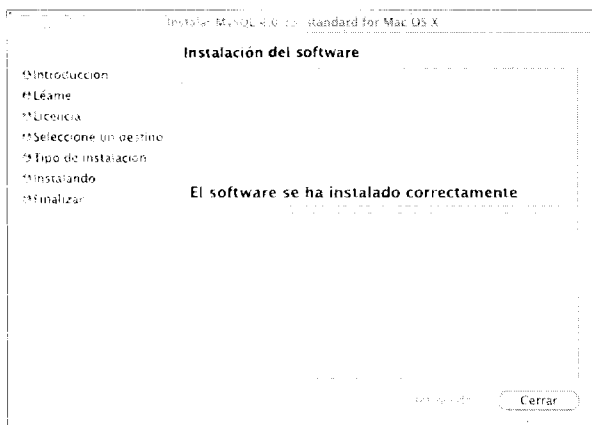


Figura 2.39.

Instalación de la base de datos MySQL (y III).

Abra de nuevo la aplicación Terminal y escriba lo siguiente en su *shell* de UNIX:

```
cd /usr/local/mysql
```

y pulse **Intro**. Posteriormente escriba:

```
sudo ./scripts/mysql_install_db
```

y pulse **Intro**. La última línea ejecuta el *script* que crea todo el sistema de tablas MySQL (figura 2.40, observe cómo el *shell* envía el mensaje "PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER!").

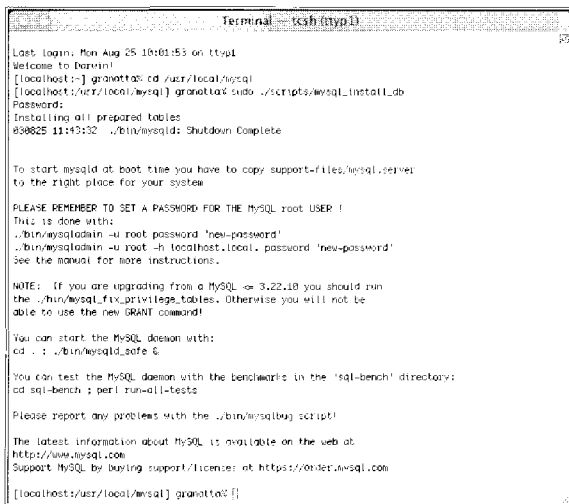


Figura 2.40.
Creación del sistema de tablas de MySQL.

Teclee entonces:

```
sudo chown -R mysql:mysql data
```

y pulse **Intro** para cambiar a *mysql* tanto el usuario como el grupo del directorio *data*.

Por último, para iniciar el funcionamiento de la base de datos MySQL teclee:

```
sudo ./bin/safe_mysqld -user=mysql &
```

En la figura 2.40 hemos visto que el *shell* de UNIX nos advertía de la conveniencia de establecer una contraseña para la cuenta de administrador (*root*). Teclee esto desde su *shell* de UNIX:

```
/usr/local/mysql/bin/mysqladmin -u  
root password 'Granatta'
```

Pulse **Intro** y trate de acceder a la cuenta *root* tecleando:

```
/usr/local/mysql/bin/mysql -uroot -p  
mysql
```

Al pulsar **Intro** se le requerirá una contraseña. Teclee aquella palabra que haya establecido como clave, y si es correcta aparecerá en su pantalla el *prompt*:

```
mysql>
```

correspondiente a la base de datos, en la cual ya puede crear tablas, registros, etc. Esto puede hacerlo manualmente, o bien instalar alguno de los entornos gráficos que permiten gestionar MySQL vía Web, como por ejemplo phpMyAdmin, del que podemos encontrar información y documentos descargables en <http://www.phpmyadmin.net>.

Instalación y puesta en marcha de phpMyAdmin

Para nuestro ejemplo hemos decidido descargar la versión 2.5.3-rc2, cuya fecha de lanzamiento es 17 de agosto de 2003 (figura 2.41).

Una vez descargado el documento comprimido a nuestro ordenador, lo descomprimimos

para obtener la carpeta phpMyAdmin-2.5.3-rc2 en nuestro Escritorio (figura 2.42). Creamos una carpeta de nombre phpMyAdmin en el directorio raíz de nuestro servidor Web; en este caso Users/granatta/Web, y posteriormente copiamos todos los ficheros de la carpeta que hemos descomprimido en el Escritorio a Users/granatta/Web/phpMyAdmin (figura 2.43).

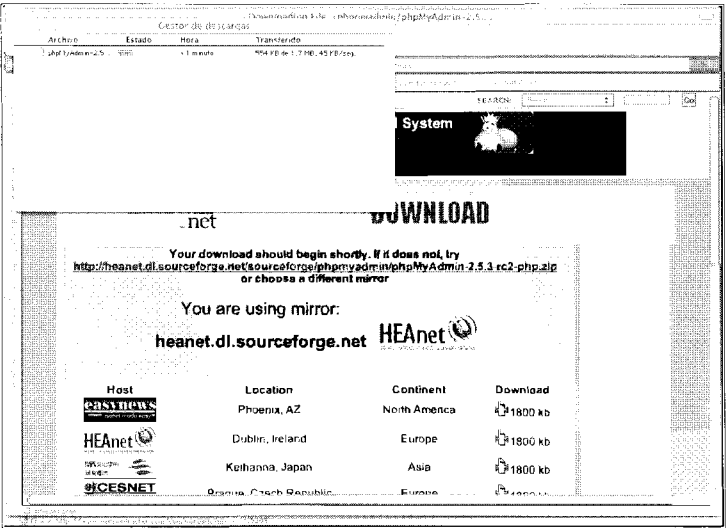


Figura 2.41.
Descarga de phpMyAdmin.

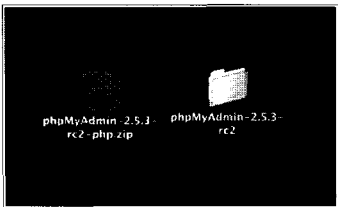


Figura 2.42.
Documentos descargados y descomprimidos.

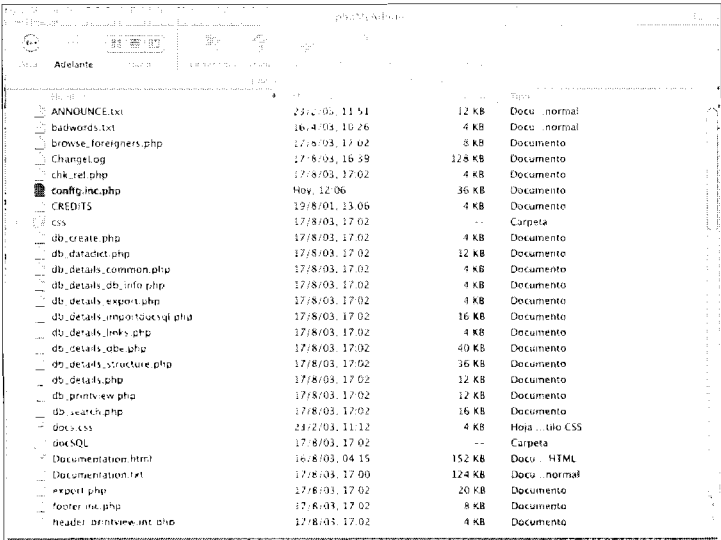


Figura 2.43.
Documentos copiados a la carpeta phpMyAdmin.

Sólo nos resta configurar phpMyAdmin para que funcione con nuestro servidor y nuestra base de datos MySQL. Para ello editaremos, con la aplicación TextEdit, el documento `config.inc.php`, que es uno de los documentos que acabamos de copiar a la carpeta `phpMyAdmin`. Una vez dentro del mismo, busque la línea:

```
$cfg['PmaAbsoluteUri'] = ''
```

en la que debe establecer la ruta absoluta en el servidor Web hasta el directorio que contiene

phpMyAdmin; por tanto, sustitúyala por la línea siguiente (figura 2.44):

```
$cfg['PmaAbsoluteUri'] = 'http://localhost.local/~granatta/phpMyAdmin/'
```

Por último, habrá de especificar el nombre del servidor Web en el que estará intalado phpMyAdmin (localhost), así como el nombre de usuario (root) y su contraseña ("Granatta", con la que accederemos a la base de datos) (figura 2.45).

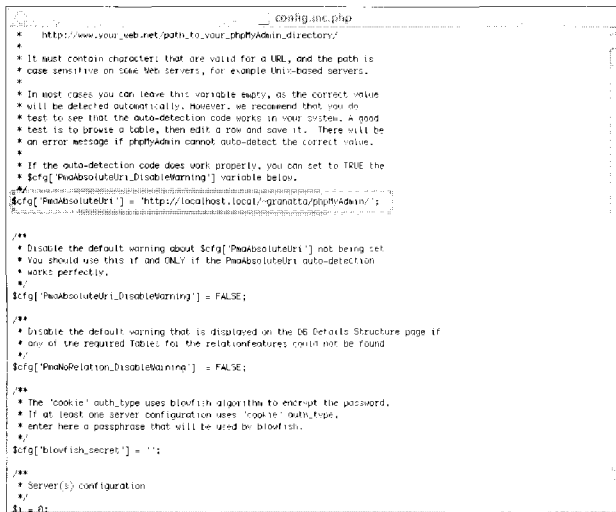


Figura 2.44.

Modificación del fichero de configuración de phpMyAdmin (I).

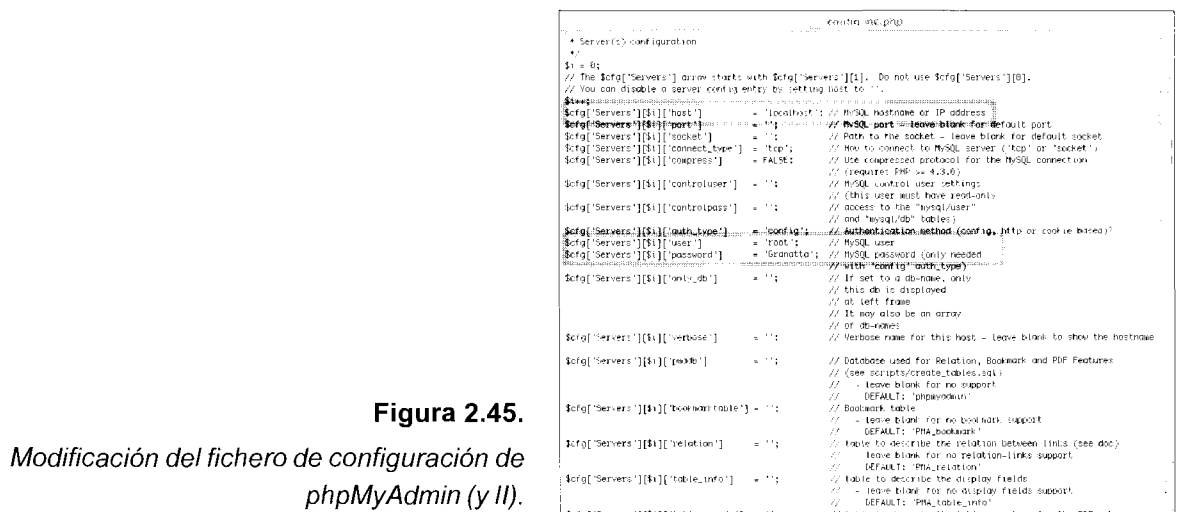


Figura 2.45.

Modificación del fichero de configuración de phpMyAdmin (y II).

Tras salvar el fichero tecleamos ahora en nuestro navegador la siguiente dirección `http://localhost.local/~granatta/phpMyAdmin`, con lo que accederemos a una pantalla de contenido igual o similar a la de la figura

2.46, comprobando que hemos instalado con éxito phpMyAdmin.

```
gotoAndPlay("capitulo_3");
```

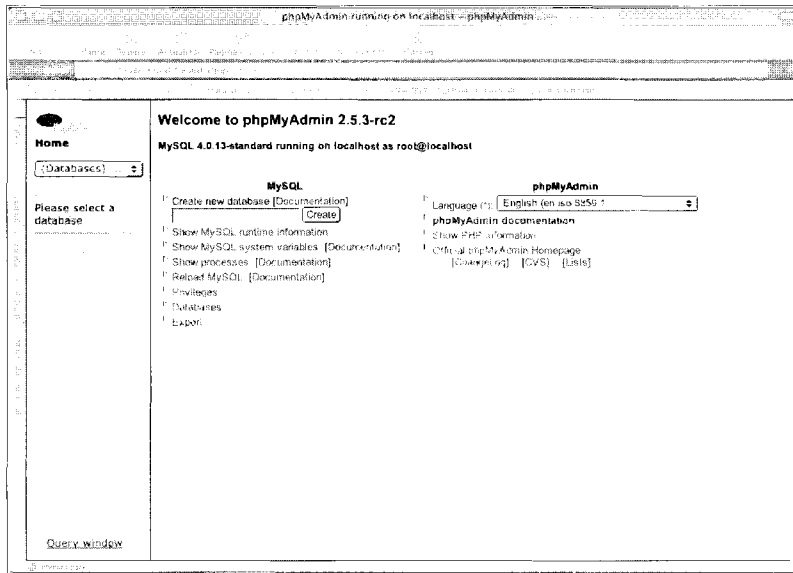


Figura 2.46.
phpMyAdmin ejecutándose con éxito en el servidor.

Capítulo 3

PHP

PHP, "hijo" del Código Abierto (*Open Source*) es, en la actualidad uno de los lenguajes de programación que cuenta con más adeptos en la Red, que crecen cada día por su sencillez a la hora de ser programado, su solidez y su constante innovación.

¿Qué es PHP?

PHP son las siglas de *PHP Hypertext Preprocessor* (Preprocesador de Hipertexto), un lenguaje interpretado de alto nivel que es insertado en documentos .html y que es ejecutado en el lado del servidor, a diferencia de otros lenguajes como JavaScript, que son interpretados en el lado del cliente (el navegador del usuario). Cuando el código de las páginas se interpreta en el servidor, el usuario recibe en su navegador el producto de esos procesos sin poder determinar qué código lo ha producido.

La forma en que PHP se inserta en los documentos se produce mediante el uso de etiquetas especiales, como `<?php`, o simplemente `<? y ?>` para indicar el comienzo y final del código PHP, respectivamente. Por ejemplo:

```
<?php
echo "hola desde PHP";
?>
```

o bien:

```
<?
echo "hola desde PHP";
?>
```

Para editar o escribir páginas o código que incluya PHP puede usar desde un editor de texto hasta un editor HTML de su preferencia.

Características del lenguaje

Como todo lenguaje, PHP dispone de sus propias especificaciones, sintaxis, variables, uso de éstas dentro y fuera de funciones, funciones predefinidas, etc. En este apartado estudiaremos aquéllas de más relevancia, intentando darle un enfoque práctico a la explicación, por lo que, antes de comenzar, cree en la raíz de su servidor una nueva carpeta con el nombre `phpBasicos`. En esta carpeta (que será `www` de *AppServ* o `htdocs` de *Apache*, dependiendo de su instalación), alojaremos los distintos documentos `.php` que iremos creando a medida que avanzamos en los contenidos de este capítulo.



Nota: Si en un determinado momento sus documentos PHP no funcionan o no lo hacen como debieran, puede encontrar los archivos correspondientes a este capítulo del libro en la carpeta `material/cap3/php/` del CD. Además, tras cada apartado de esta sección que tenga disponible un archivo de práctica, encontrará referenciado el nombre con el que puede encontrarlo en dicha carpeta.

Características generales de PHP

Cuando escribimos código en lenguaje PHP es muy importante recordar que al finalizar cada sentencia ha de agregarse un punto y coma:

```
$miVariable = 5;
```

Esta expresión es incorrecta y generará un error. En cambio, la expresión:

```
$miVariable = 5;
```

sí es correcta. A medida que escriba sus scripts de PHP, puede que desee añadir comentarios para una mejor comprensión del código. Esto puede realizarlo anteponiendo la cadena de texto `//` al comentario si éste es de una sola línea o bien colocarlo entre los caracteres `/*` y `*/` si ocupan más de una línea:

```
// esto es un comentario de una sola línea
```

```
/*  
esto es  
un comentario  
de varias líneas  
*/
```

Impresión de datos

Por "impresión de datos" hacemos referencia a los datos que, mediante PHP, mostramos por pantalla, y no a un resultado impreso utilizando un dispositivo externo. En PHP existen varias formas de impresión de datos, siendo la más utilizada aquella en la que se utiliza la función `echo`, cuya sintaxis es la siguiente:

```
<?  
echo "hola mundo";  
?>
```

Este código nos mostrará la frase "hola mundo" en la pantalla. Escriba el código utilizando el editor de texto que desee y guárdelo con el nombre `holaMundo.php`. Posteriormente, ábralo en su navegador tecleando la dirección `http://localhost/flashphp/phpBasicos/holaMundo.php` para comprobar que el texto que se muestra en su navegador es:

```
hola mundo
```

tal y como puede ver en la figura 3.1. Puede encontrar el documento `holaMundo.php` en la carpeta `material/capitulo3/php/` del CD.

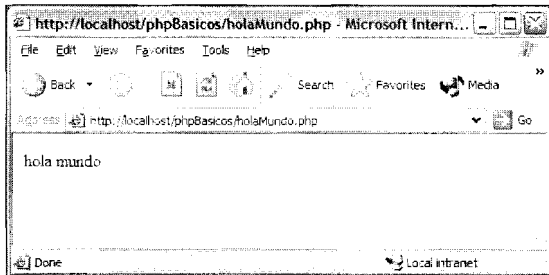


Figura 3.1.

Impresión de datos usando PHP.

Utilización de variables en PHP

Para crear variables en PHP, anteponga el carácter `$` al nombre de la variable que usted haya elegido:

```
$miVariable  
$GET['miVariable']
```

PHP es un lenguaje *case sensitive*; esto es, sensible a la diferencia entre mayúsculas y minúsculas, por lo habrá de prestar atención puesto que, por ejemplo, `$miVariable` y `$mivARIABLE` son dos variables distintas.

Variables predefinidas

PHP proporciona una enorme cantidad de variables predefinidas que están disponibles para cualquier *script* de código que se ejecute. Muchas de estas variables dependen de la configuración del servidor, pudiendo encon-

trar las que vamos a utilizar dentro de matrices, variables almacenadas en *cookies* y sesiones o enviadas vía POST o GET. La sintaxis para acceder a estas variables que se encuentran dentro de las matrices es la siguiente:

```
$_GET['miVariable'];
```

Donde `miVariable` es una variable pasada a un *script* en forma GET.

```
$_POST['miVariable'];
```

Donde `miVariable` es una variable pasada a un *script* en forma POST.

```
$_COOKIE['miVariable'];
```

Donde `miVariable` es una variable almacenada en una *cookie*.

```
$_SESSION['miVariable'];
```

Donde `miVariable` es una variable almacenada en una sesión de usuario.

```
$_GLOBALS['miVariable'];
```

Donde `miVariable` es una variable almacenada por nosotros en la matriz de variables globales.

Variables locales, globales y superglobales

Podemos clasificar las variables en PHP en función del lugar en el que se utilizan. Por ejemplo, cuando estamos utilizando funciones, las variables que declaramos dentro de las mismas se conocen como variables locales y son eliminadas o borradas cuando la función termina de ejecutarse:

```
function sumar ($n,$m){
    $resultado = $n + $m;
    return $resultado;
}
echo sumar(5,6);
```

En este ejemplo la variable `$resultado` es local para la función `sumar`.

Si desea utilizar variables externas a la función, ha de hacerlas globales previamente, para informar a la función de que va a necesitar valores que están alojados fuera de ella:

```
$n = 5;
function sumar ($m){
    global $n;
    $resultado = $n + $m;
    return $resultado;
}
echo sumar(6);
```

En este otro ejemplo, la función `sumar` toma el valor de una variable externa (`$n`) para realizar las operaciones correspondientes.

Las variables autoglobales o superglobales son aquellas para las que no es necesario hacer referencia en el interior de la función, puesto que dicho proceso se ha realizado ya de una forma que podríamos denominar "automática". Estas variables son `$_GET`, `$_POST`, `$_COOKIE`, `$_SESSION` y `$_GLOBALS`.

```
$_GLOBALS['n'] = 5;
function sumar ($m){
    $resultado = $n + $m;
    return $resultado;
}
echo sumar(6);
```

En la dirección Web <http://www.php.net/manual/es> tiene disponible la última versión del manual de PHP, donde puede encontrar más información acerca del modo en que se manejan las variables en este lenguaje.

Concatenación de cadenas de texto

Para concatenar dos cadenas de texto en PHP se utiliza el carácter `.`, tal y como se muestra en el siguiente ejemplo:

```
<?
$variable1 = "hola";
$variable2 = $variable1 . " mundo";
echo $variable1 . "<br>".$variable2;
?>
```

Mediante este *script* asignamos como contenido de `$variable1` la cadena de texto "hola", que concatenamos con " mundo" para generar el contenido de `$variable2`, "hola mundo".

PHP es un lenguaje capaz de acceder al valor de las variables que se encuentran en el interior de las cadenas de texto delimitadas por comillas dobles, por lo que las expresiones:

```
echo $variable1 . "<br>".$variable2;
```

y:

```
echo "$variable1 <br> $variable2";
```

son equivalentes. Escriba el *script* anterior en su editor de texto y guárdelo como `concatenacionCadenas.php` para posteriormente probarlo en su navegador tecleando <http://localhost/flashphp/phpBasicos/concatenacionCadenas.php>.

Tal y como puede ver en la figura 3.2, el texto que aparece es:

```
hola
hola mundo
```

ya que, además de los contenidos de las dos variables, el *script* contiene el texto `"
"`, que es un *tag* HTML utilizado para generar un salto de línea. Puede encontrar el documento `concatenacionCadenas.php` en la carpeta `material/capitulo3/php/` del CD.

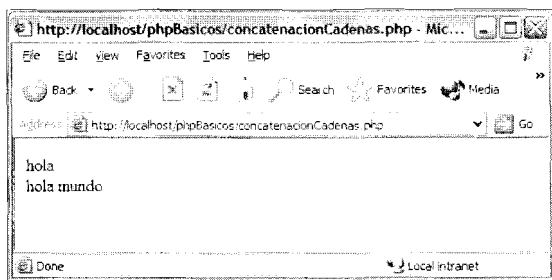


Figura 3.2.

Concatenación de cadenas de texto.

Adición de más texto a una variable ya existente

Si desea agregar más información a una variable ya existente que además ya dispone de contenido, puede usar igualmente el carácter `..`:

```
<?
$variable1 = "esto es algo de
texto";
echo $variable1."<br>";
$variable1 .= " ahora variable1
también tiene este texto";
echo $variable1;
?>
```

En este ejemplo, la expresión:

```
$variable1.=" ahora variable1
también tiene este texto";
```

es equivalente a:

```
$variable1 = $variable1 . "ahora
variable1 también tiene este texto";
```

por lo que finalmente el contenido de la variable `$variable1` es "esto es algo de texto ahora variable1 también tiene este texto", lo que puede comprobar copiando el código en su editor de texto, guardarlo con el nombre que desee y añadirlo a la carpeta `phpBasi-`

`cos` de su servidor para poder posteriormente abrirlo en su navegador y ver el resultado.

Utilización y manipulación de arrays

Al igual que vimos en el capítulo dedicado a Flash, las variables tienen el inconveniente de que sólo pueden almacenar un valor en un momento determinado, por lo que PHP también dispone de *arrays* como estructura para almacenar datos. Para crear un nuevo *array* en PHP, usaremos la palabra reservada *array* siguiendo la sintaxis:

```
$miArray = array(elemento1,
elemento2, ..., elementoN);
```

Por ejemplo:

```
$miArray = array(0,1,2,3,4,5,6);
```

sirve para crear un *array* de siete elementos, en el que el acceso a cada posición se realiza utilizando un índice cuyo valor está entre 0 y el número de elementos menos uno, es decir, entre 0 y 6, por lo que `$miArray[6]` almacena el valor 6.

Sin embargo, también se pueden crear *arrays* en los que el acceso a cada posición se realice mediante una cadena de texto, como por ejemplo:

```
$miArray = array("nombre"=> "Juan",
"apellido"=> "Pérez");
```

donde los contenidos del *array* ("Juan" y "Pérez") se referencian por `$miArray["nombre"]` y `$miArray["apellido"]`, respectivamente.

Para añadir más elementos al final del *array* utilice la función `array_push` siguiendo la sintaxis:

```
array_push(nombre_del_array,  
nuevo_valor1, ..., nuevo_valorN);
```

Por ejemplo, para añadir dos nuevos elementos al final del *array* `$miArray` utilizaríamos:

```
array_push($miArray,7,8);
```

Recorriendo los elementos de un *array*

Utilizaremos bucles para recorrer un *array*, extrayendo los valores que tiene almacenados mediante uno de los dos métodos a nuestra disposición. El primero de ellos consiste en utilizar como índice el valor de una variable que se incrementa en cada iteración del bucle, mientras que el segundo consiste en utilizar funciones predefinidas de PHP (`each`, `next`), que extraen los valores mientras incrementan el apuntador del *array*.



Nota: El apuntador de un *array* en PHP almacena la posición del registro siguiente al último del que se extrajeron datos.

Para entender mejor qué son los apuntadores y cómo funcionan, vamos a crear un archivo en el que definiremos un *array*, del que extraeremos los datos primeramente mediante un bucle `for` y una variable índice, y posteriormente mediante una función predefinida.

El cómo recorrer un *array* utilizando un bucle `for` puede encontrarlo en el documento `material/capitulo3/php/recorrerArregloFor.php` del CD:

```
<?  
for( $n=0; $n < sizeof($myArray);  
$n++){  
    echo $n. " : ".$myArray[$n]. "<br>";  
}  
echo "<br>segunda vez con for<br>---<br>";  
  
for( $n=0; $n < sizeof($myArray);  
$n++){  
    echo $n. " : ".$myArray[$n]. "<br>";  
}  
?>
```

Teclee el código en un archivo nuevo con su editor de texto, guárdelo como `recogerArregloFor.php` y ábralo en su navegador tecleando `http://localhost/flashphp/phpBasicos/recogerArregloFor.php`. La información que debe aparecer en su pantalla ha de ser la misma que se muestra en la figura 3.3:

```
0:1  
1:2  
2:3  
3:4  
4:5  
5:6  
6:7
```

```
segunda vez con for  
----  
0:1  
1:2  
2:3  
3:4  
4:5  
5:6  
6:7
```

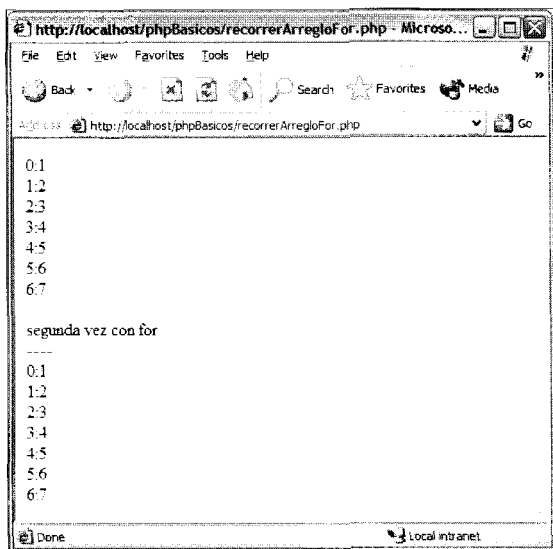


Figura 3.3.

Extrayendo datos de un array usando un bucle for.

En el ejemplo anterior, hemos extraído en dos ocasiones los valores almacenados en el *array* para mostrar que el apuntador no fue alterado tras la primera vez.

Ahora vamos a recorrer el *array* usando *each*, pero antes vamos a analizar esta función; *each* devuelve dos pares de valores en un *array* cuyos campos pueden accederse de dos formas, usando los índices 0 y 1 o bien las claves *key* y *value*. Este *array* suele asignarse a una variable que es la que se maneja para extraer los datos.

Por ejemplo, si creamos la siguiente variable:

```
$temp = each($myArray);
```

temp dispondrá de los siguientes valores:

```
$temp[0]; (índice)
$temp[1]; (valor)
$temp['key']; (índice)
$temp['value']; (valor)
```

Copie el siguiente código en un nuevo documento de su editor de texto y guárdelo con el nombre *recogerArregloEach.php*:

```
<?
$myArray = array(1,2,3,4,5,6,7);
while($temp = each($myArray)){
    echo $temp['key'].
    ":".$temp['value']."<br>";
}

echo "<br>segunda vez con each<br>--
--<br>";
while($temp = each($myArray)){
    echo $temp['key'].
    ":".$temp['value']."<br>";
}
?>
```

Abra el documento en su navegador tecleando <http://localhost/flashphp/phpBasicos/recogerArregloEach.php>.

La información que debe aparecer en su pantalla ha de ser la misma que se muestra en la figura 3.4:

```
0:1
1:2
2:3
3:4
4:5
5:6
6:7

segunda vez con each
----
```

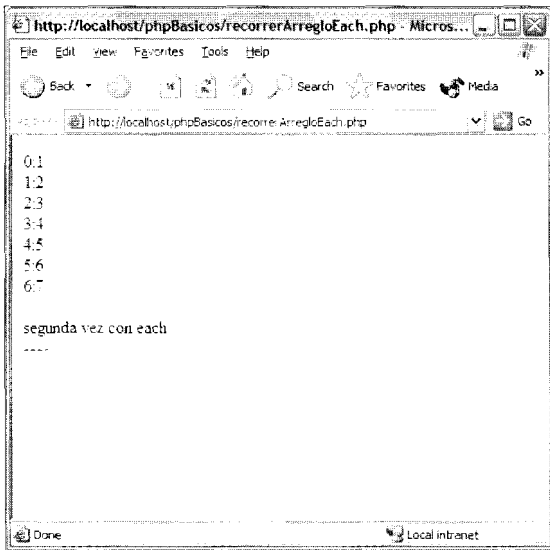


Figura 3.4.
Extrayendo datos de un array usando each.

En este caso sólo se muestra el contenido del *array* una vez, ya que al recorrerlo la primera vez se iba modificando el apuntador. Al intentar recorrer la estructura por segunda vez, el apuntador ya se encontraba al final del *array*, por lo que no se encuentra ningún contenido que mostrar. Puede encontrar el código de este *script* en el documento `material/capitulo3/php/recorrerArregloEach.php` del CD.

La solución a este inconveniente nos la proporciona la función `reset`, que sirve para reiniciar la posición del apuntador mediante la sintaxis:

```
reset($nombre_del_array);
```

Cree un documento de nombre `recorrerArregloEachReset.php` y teclee de nuevo el código (puede encontrarlo en el documento `material/capitulo3/php/recorrerArregloEachReset.php`), añadiendo la línea que contiene la función `reset` antes de recorrer el *array* por segunda vez:

```
<?
$myArray = array(1,2,3,4,5,6,7);
while($temp = each($myArray)){
    echo $temp['key'].
    ":".$temp['value']."<br>";
}

echo "<br>segunda vez con each<br>--<br>";

reset($myArray);

while($temp = each($myArray)){
    echo $temp['key'].
    ":".$temp['value']."<br>";
}
?>
```

Abra el documento en su navegador tecleando `http://localhost/flashphp/phpBasicos/recogerArregloEachReset.php`.

La información que debe aparecer en su pantalla ha de ser la misma que se muestra en la figura 3.5:

```
0:1
1:2
2:3
3:4
4:5
5:6
6:7
```

segunda vez con each

```
0:1
1:2
2:3
3:4
4:5
5:6
6:7
```

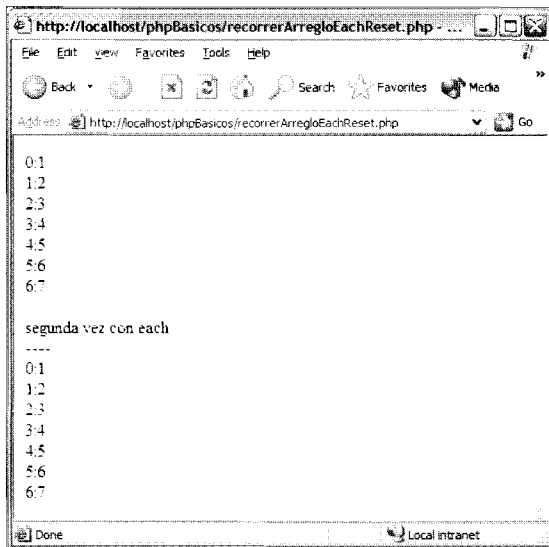


Figura 3.5.

Extrayendo datos de un array usando each y reiniciando el apuntador.

Existe una alternativa a la de utilizar las claves key y value, y es la de asignar el

resultado a variables que definiremos en una lista:

```
<?
$myArray = array(1,2,3,4,5,6,7);
while(list ($clave, $valor) =
each($myArray)){
    echo "$clave : $valor <br>";
}
?>
```

Copie el código en un nuevo documento de su editor de texto y guárdelo con el nombre `recorrerArregloEachList.php`, para posteriormente abrirlo en su navegador tecleando `http://localhost/flashphp/phpBasicos/recorrerArregloEachList.php` (este código se encuentra disponible en el documento material/capitulo3/php/recorrerArregloEachList.php del CD).

La información que debe aparecer en su pantalla ha de ser la misma que se muestra en la figura 3.6:

```
0:1
1:2
2:3
3:4
4:5
5:6
6:7
```

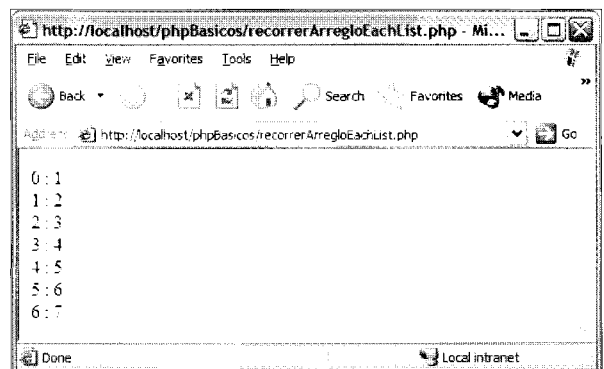


Figura 3.6.

Extrayendo datos de un array usando each y list.

Transformaciones de cadenas de texto en arrays y viceversa

En numerosas ocasiones habrá de extraer fragmentos de cadenas de texto para convertirlas en números, y viceversa, como puede ocurrirle cada vez que haya de utilizar fechas habiendo extraído la información de una base de datos. En estos casos, el formato de los datos puede ser *aaaa-mm-dd* (por ejemplo, "2003-07-25"), y tal vez usted desee trabajar con el mes. También puede ocurrir que disponga de un *array* y quiera convertirlo en una cadena, uniendo los elementos contenidos con algún carácter que usted determine.

Las funciones que vamos a utilizar para realizar las conversiones serán `explode` (para convertir un *array* en una cadena de texto) e `implode` (para convertir una cadena de texto en un *array*). Veamos un ejemplo de uso de cada una de ellas. La sintaxis para usar la función `explode` es la siguiente:

```
$nombre_array=explode(separador,
cadena_texto);
```

Puede encontrar un ejemplo en el documento `material/capitulo3/php/explode.php` del CD, en el que encontrará el siguiente código:

```
<?php
$fecha="2003-05-25";
$arrayFecha = explode("-", $fecha);
echo "el mes es " . $arrayFecha[1];
//el mes se encuentra en el índice 1
?>
```

Este código divide la cadena de texto contenida en la variable `$fecha` por el carácter especificado como `separador` (" - "), con lo que se obtienen tres elementos (2003, 05 y 25) que se almacenan en el *array* `$arrayFecha`. Poste-

riormente se muestra por pantalla el contenido de la posición 1 del *array*, en la que se ha almacenado el texto correspondiente al mes, por lo que si crea un documento PHP de nombre `explode.php` con este código para abrirlo posteriormente en su navegador, debe obtener el mensaje:

```
el mes es 05
```

Puede encontrar el documento `explode.php` en la carpeta `material/capitulo3/php/` del CD.

La sintaxis para usar la función `implode` es la siguiente:

```
$cadena=implode(separador,
&nombre_array);
```

Por ejemplo:

```
<?php
$arrayFecha = array
("2003", "05", "25");
$fecha = implode("-", $arrayFecha);
echo "esta es la fecha " . $fecha;
?>
```

El código, que puede encontrar en el documento `material/capitulo3/php/implode.php` del CD, realiza el proceso inverso al del ejemplo de uso de `explode`. Disponemos de un *array* con tres posiciones cuyos contenidos vamos a concatenar mediante el carácter especificado en el `separador`, por lo que generaremos una nueva cadena de texto almacenada en la variable `$fecha`. Copie el código en un nuevo documento de su editor de texto y guárdelo con el nombre de `implode.php` para abrirlo con su navegador, en el que obtendrá el mensaje:

```
esta es la fecha 2003-05-25
```

Función para manejar fechas (date)

La función `date` retorna una cadena de texto con un formato que se asigna según los parámetros que se especifiquen y que pueden ser los mostrados en la tabla 3.1.

Tabla 3.1.

Parámetros para formatear las cadenas de texto generadas con la función `date`.

Parámetro	Descripción
<i>a</i>	"am" o "pm"
<i>A</i>	"AM" o "PM"
<i>d</i>	Día del mes, usando dos dígitos con un cero a la izquierda, desde "01" a "31".
<i>D</i>	Día de la semana, usando un texto de tres letras, por ejemplo "Fri" para indicar <i>Viernes</i> .
<i>F</i>	Mes del año, usando un texto, por ejemplo "January" para indicar <i>Enero</i> .
<i>h</i>	Hora, usando dos dígitos con un cero a la izquierda, desde "01" a "12".
<i>H</i>	Hora, usando dos dígitos con un cero a la izquierda, desde "00" a "23".
<i>g</i>	Hora, sin ceros a la izquierda, desde "1" a "12".
<i>G</i>	Hora, sin ceros a la izquierda, desde "0" a "23".

Parámetro	Descripción
<i>i</i>	Número de minutos, usando dos dígitos con un cero a la izquierda, desde "00" a "59".
<i>j</i>	Día del mes, sin cero a la izquierda, desde "1" a "31".
<i>I</i>	Día de la semana, usando un texto, por ejemplo "Friday" para indicar <i>Viernes</i> .
<i>L</i>	"1" o "0", para indicar si el año es bisiesto o no.
<i>m</i>	Mes del año, usando dos dígitos con un cero a la izquierda, desde "01" a "12".
<i>n</i>	Mes del año, sin cero a la izquierda, desde "1" a "12".
<i>M</i>	Mes del año, usando un texto de tres letras, por ejemplo "Jan" para indicar <i>Enero</i> .
<i>s</i>	Número de segundos, usando dos dígitos con un cero a la izquierda, desde "00" a "59".
<i>S</i>	Sufijo ordinal en idioma inglés, usando un texto de dos caracteres, por ejemplo "th" o "nd".
<i>t</i>	Número de días del mes especificado, usando dos dígitos, desde "28" a "31".
<i>U</i>	Número de segundos desde el valor de la fecha <i>Epoch</i> (las 00:00:00 horas del día 01-01-1970).

Parámetro	Descripción
w	Día de la semana, usando un dígito, desde "0" (<i>Domingo</i>) a "6" (<i>Sábado</i>).
Y	Año, en formato de cuatro cifras, por ejemplo "1999".
Y	Año, en formato de dos cifras, por ejemplo "99".
Z	Día del año, desde "0" a "365".
Z	Diferencia horaria en segundos (de "-43200" a "43200").

Por ejemplo:

```
<?php
echo date("Y-m-d h:m:s a");
?>
```

Copie este código en un nuevo archivo y guárdelo con el nombre `fechaSimple.php`. Cuando lo abra en su navegador, verá en su pantalla la fecha en el formato `aaaa-mm-dd hh:mm:ss am/pm`. Puede encontrar el documento `fechaSimple.php` en la carpeta `material/capitulo3/php/` del CD.

Imprimir la fecha en castellano

Cuando se desea escribir la fecha en castellano hay que tener en cuenta varios aspectos.

Evitar que partes del texto se interpreten como parámetros

Si va utilizar alguna letra de las anteriormente especificadas en la tabla 3.1 dentro de una palabra, ha de anteponerle el carácter `\`, para

que no sea interpretada; es decir, si deseamos mostrar la fecha en un formato como el siguiente:

```
"27 de julio, 2003, 5:00:00 am"
```

habríamos de escribir el siguiente código PHP:

```
<?php
echo date("d \de F, Y, h:m:s a");
?>
```

donde el anteponer el carácter `\` a la `d` de `de` sirve para indicarle a PHP que no interprete la segunda letra `d` como un parámetro para la fecha, puesto que dicha letra pertenece a la cadena de texto `de`. En la carpeta `material/capitulo3/php/` del CD encontrará el documento `fechaEscape.php` con este *script*. Pruébelo en su navegador para leer la fecha en castellano.

Convertir los nombres de meses y días a castellano

Los nombres para los meses están disponibles en inglés, así que disponemos de varias alternativas si queremos mostrarlos en otro idioma. Una de las más comunes es la de crear un *array* con los nombres de los meses en castellano sabiendo que *Enero* es el mes número 1, con lo cual, como los *arrays* se comienzan a numerar desde el índice 0, no utilizaremos el primer campo del *array* de meses que creemos:

```
$meses = array(null, "Enero",
"Febrero", "Marzo", "Abril", "Mayo",
"Junio", "Julio", "Agosto",
"Septiembre", "Octubre",
"Noviembre", "Diciembre");
```

El siguiente paso sería el de obtener el número del mes y utilizarlo como índice en nuestro

array para así obtener el nombre de dicho mes en castellano.

Utilizando `date("n")` obtendremos un número que utilizaremos como índice en el *array* `$meses`, y así conseguiremos el nombre del mes en castellano, que almacenamos en la variable `$mes`:

```
$mes = $meses[date("n")];
```

Para obtener los nombres de los días operaremos de forma similar, con la diferencia de que el *array* `$dias` ha de comenzar en *Domingo*, siendo este día el que tenga asociado el número de índice 0:

```
$dias = array  
("Domingo", "Lunes", "Martes", "Miércoles", "Jueves",  
"Viernes", "Sábado");
```

Utilizamos `date("w")` para obtener el índice del día actual, obteniendo un número entre 0 y 6, que usamos como índice en el *array* `$dias` para obtener el nombre del día en castellano, almacenándolo en la variable `$dia`.

Para obtener la fecha en el formato "Domingo 27 de julio, 2003, 5:00:00 am" utilizando el nombre del mes (almacenado en `$mes`) y el nombre del día (almacenado en `$dia`), hemos de dividir la fecha en varias partes:

```
Valor de variable $dia + espacio +  
"día del mes" + "de" + Valor de variable $mes + espacio + ", año, hh:mm:ss  
am/pm"
```

Los espacios podemos añadirlos a las variables o escribirlos por separado. Añadiéndolos a las variables se utilizaría:

```
echo "$dia "
```

Para escribirlos por separado utilizaríamos:

```
echo $dia . " " .
```

Reuniendo todo el código que hemos escrito hasta ahora (y que puede encontrar en el documento `material/capitulo3/php/fechaEsp.php` del CD), obtenemos este *script*:

```
<?php  
$meses =  
array(0,"Enero","Febrero","Marzo","Abril","Mayo",  
"Junio","Julio","Agosto","Septiembre",  
"Octubre","Noviembre","Diciembre");  
$dias = array  
("Domingo","Lunes","Martes","Miércoles",  
"Jueves","Viernes","Sábado");  
$mes = $meses[date("n")];  
$dia = $dias[date("w")];  
echo "$dia " . date("j"). " de $mes  
" . date(" Y, h:m:s a");  
?>
```

Copie el código en un nuevo documento de su editor de texto, guárdelo con el nombre `fechaEsp.php` y ábralo en su navegador tecleando `http://localhost/flashphp/phpBasicos/fechaEsp.php`. El resultado debe ser similar al que se muestra en la figura 3.7.

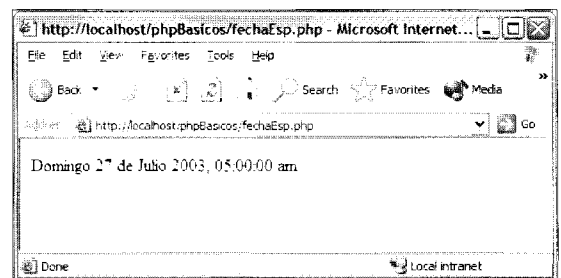


Figura 3.7.

Impresión de la fecha en castellano.

Impresión de fechas pasadas o futuras

Para imprimir fechas pasadas o futuras podemos utilizar la función `mktime`, que se pasa como parámetro a la función `date`. El formato de `mktime` es el siguiente:

```
int mktime ( int hora, int minutos,
int segundos, int mes, int día, int
año );
```

```
<?php
echo "Fin de año ";
echo date("Y-m-
d",mktime(0,0,0,12,31,2003));
?>
```

Este código se corresponde con el del documento `material/capitulo3/php/fechaMktime.php` del CD. Para generar una fecha futura en base a la fecha actual puede utilizar la función `date` para obtener los datos. Por ejemplo, vamos a hacer uso de `mktime` para obtener el día siguiente (mañana):

```
<?php
$manana = mktime(0,0,0,date("m")
,date("d")+1,date("Y"));
echo date("Y-m-d",$manana);
?>
```

En el código anterior, que encontrará en el documento `material/capitulo3/php/fechaMktimeManana.php` del CD, puede ver cómo la expresión `date("m")` retorna el mes actual, la expresión `date("d")+1` retorna el día siguiente al actual, y la expresión `date("Y")` retorna el año en curso.

Manipulación de ficheros

Las funciones más comunes para trabajar con ficheros son las de abrir, leer, escribir y cerrar, lo que veremos con detalle en este apartado. El proceso normal para trabajar con un fichero es el siguiente:

- Abrir el fichero.
- Leer datos del fichero / Escribir datos en el fichero.
- Cerrar el fichero.

Abrir un fichero

Cuando trate de abrir un fichero, pueden ocurrir dos cosas, que exista y se abra normalmente, o bien que no exista, con lo que PHP tratará de crear un nuevo archivo con el nombre del fichero que usted deseaba abrir. La función que se utiliza para abrir/crear archivos es `fopen` y su sintaxis es:

```
fopen(nombre_fichero, modo, usar
include_path);
```

Esta función retorna un valor booleano `false` si no es capaz de abrir/crear el fichero. El primer parámetro (`nombre_fichero`) contiene la dirección absoluta o relativa del directorio en el que se encuentra el fichero que desea abrir/crear. Si es usted usuario de Windows, recuerde que a la hora de utilizar rutas absolutas es necesario que añada el carácter `\` antes de cada uno de los caracteres `\` que la ruta del fichero contenga. Por ejemplo:

```
"C:\carpeta\archivos\fichero.txt";
```

La anterior declaración de la ruta es incorrecta, al contrario que la que mostramos a continuación:

```
"C:\\carpeta\\archivos\\fichero.txt";
```

Los posibles valores para el parámetro `modo` puede observarlos en la tabla 3.2:

Tabla 3.2.
Valores para el parámetro modo cuando se desea abrir/crear ficheros.

Parámetro	Descripción
"r"	Abre el fichero para sólo lectura situando su apuntador al comienzo del mismo.
"r+"	Abre el fichero para lectura y escritura situando su apuntador al comienzo del mismo.
"w"	Abre el fichero para sólo escritura situando su apuntador al comienzo del mismo y elimina los contenidos que el fichero tuviera almacenados. Si el fichero no existe, trata de crearlo.
"w+"	Abre el fichero para lectura y escritura situando su apuntador al comienzo del mismo y elimina los contenidos que el fichero tuviera almacenados. Si el fichero no existe, trata de crearlo.
"a"	Abre el fichero sólo para añadir contenido situando su apuntador al final del mismo. Si el fichero no existe, trata de crearlo.

Parámetro	Descripción
"a+"	Abre el fichero para lectura y escritura (añadiendo contenido) situando su apuntador al final del mismo. Si el fichero no existe, trata de crearlo.

El tercer parámetro se utiliza cuando hay una ruta asignada en la variable `include_path` y se quiere utilizar ésta. El resultado de la apertura de un fichero es asignado a una variable que permitirá referenciar el fichero con posterioridad. Algunos ejemplos de apertura de ficheros son:

```
$fichero = fopen("/www-root/pepe/datos.txt", "r");  
$fichero = fopen("http://localhost/fichero.txt", "r");
```

Lectura de un fichero

Una vez que hemos abierto el fichero, podemos leer sus contenidos usando la función `fread`, cuya sintaxis es:

```
fread(referencia, largo);
```

Donde `referencia` es la variable que contiene el resultado obtenido con `fopen`, y sirve para referenciar el fichero, mientras que `largo` es la cantidad en bytes que deseamos leer. `fread` retorna los resultados leídos hasta que alcance el número de bytes indicado en `largo` o hasta que se alcance el final del fichero. Lo más común es establecer como valor para el parámetro `largo` la cantidad total en bytes del archivo, que se obtiene mediante la función `filesize`, cuya sintaxis es:

```
filesize($ruta_completa_al_fichero);
```

Ejemplo de lectura de un fichero:

```
<?
$ruta = "fichero1.txt";
$fp = fopen($ruta,"r");
$datos = fread($fp,
filesize($ruta));
fclose($fp);
echo $datos;
?>
```

Este código puede encontrarlo en el documento `material/capitulo3/php/leerFichero.php` del CD. Si lo abre en su navegador, podrá observar cómo se imprime en la pantalla el contenido del documento `fichero1.txt`.

También puede leer directamente los contenidos de un fichero mediante la función `file`, cuya sintaxis es:

```
file ($fichero,include_path);
```

La diferencia de este modo de acceder a la información con respecto a la anterior radica en que `file` retorna el contenido del fichero como un *array*, en el que cada uno de los elementos se corresponde con cada una de las líneas del fichero.

Escribir en un fichero

Una vez que está abierto el fichero, podemos escribir en él. Para tal tarea haremos uso de las funciones `fwrite` o `fputs`, cuyo funcionamiento es el mismo:

```
fwrite(referencia, cadena, largo);
```

donde `referencia` es una variable que referencia al fichero, `cadena` es el contenido que deseamos escribir en el fichero y `largo` es un parámetro opcional que se corresponde

con el peso en bytes. Si utilizamos este último parámetro, la escritura se realizará hasta el punto especificado en `largo` o cuando la cadena se finalice. Por ejemplo:

```
<?
$datos = "Hola de nuevo \n";
$ruta = "fichero2.txt";
$fp = fopen($ruta,"a+");
fwrite($fp, $datos);
fclose($fp);
?>
```

Este código, que puede encontrar en el documento `material/capitulo3/php/escribirFichero.php` agrega la línea "Hola de nuevo" al documento `fichero2.txt` cada vez que es llamado.

Cerrar un fichero

Tal y como habrá podido observar en los ejemplos anteriores, para cerrar un fichero se utiliza la función `fclose`, pasando la referencia al fichero como parámetro:

```
fclose(referencia);
```

Ejemplo práctico: contador basado en fichero de texto

Como ejemplo práctico de los conocimientos adquiridos en este apartado, vamos a utilizar el ejemplo por excelencia de uso de manipulación de ficheros a través del tiempo, la creación de un contador de visitas que escriba datos en un archivo de texto y despliegue el número de impresiones que se han obtenido. Esta tarea puede descomponerse en seis etapas:

- Abrir el fichero.
- Leer el contenido.

- Incrementar en una unidad el valor leído.
- Cerrar el fichero.
- Escribir en el fichero el nuevo dato.
- Imprimir el nuevo dato.

```
<?
// Especificamos la ruta del fichero que contiene la cantidad de impresiones
$ruta = "contador.txt";
// Abrimos el fichero para lectura
$fp = fopen($ruta,"r+");
// Leemos su información
$datos = fread($fp,filesize($ruta));
// Lo cerramos
fclose($fp);
// Aumentamos en una unidad la cantidad de impresiones almacenadas
$datos++;
// Abrimos el fichero para escritura
$fp = fopen($ruta,"w+");
// Escribimos el nuevo dato
fputs($fp, $datos);
// Cerramos de nuevo el fichero
fclose($fp);
// Imprimimos en pantalla la cantidad de impresiones.
echo "hits=" . $datos;
?>
```

Este código puede encontrarlo en el documento `material/capitulo3/php/contador.php` del CD. Ábralo en su navegador para comprobar cómo se modifica el valor contenido en el documento `contador.txt` cada vez que accede al documento `.php`.

Envío de correos utilizando PHP

`mail` es una de las funciones más útiles y sencillas de que dispone PHP. Su utilidad es la de que permite enviar correos y su sintaxis es la siguiente:

```
mail (destinatario, asunto,
mensaje,cabeceras_adicionales);
```

`destinatario` es la dirección de correo electrónico a la que se quiere enviar el mensaje, `asunto` se corresponde con el *subject* o tema del correo y `mensaje` es el cuerpo del correo. Un ejemplo sencillo de envío:

```
mail("pepe@pepe.net", "Sobre este
tema", "Línea 1\nLínea 2\n");
```

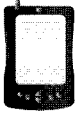
Como parámetro adicional se cuenta con lo que se denominan *cabeceras*, en las que se pueden incluir el remitente del correo, una dirección para la respuesta o el formato del correo (texto plano o formato HTML).

Si, por ejemplo, desea enviar un correo con varias cabeceras puede utilizar:

```
mail("pepe@pepe.net", "Sobre este tema", "Línea 1\nLínea 2\n",  
"From: webmaster@$SERVER_NAME\nReply-To: webmaster@$SERVER_NAME\nX-Mailer: PHP/"  
" . phpversion());
```

Si, además de varias cabeceras, también quiere especificar formato HTML:

```
mail("pepe@pepe.net", "Sobre este tema", "Línea 1\nLínea 2\n",  
"From: webmaster@$SERVER_NAME\nReply-To: webmaster@$SERVER_NAME\n Content-Type:  
text/html; charset=iso-8859-15\nX-Mailer: PHP/" . phpversion());
```



Nota: La línea Content-Type: text/html; charset=iso-8859-15\n es la que indica el formato HTML junto con una selección de caracteres para idiomas que disponen de acentos.

Más información sobre PHP

Para más información sobre PHP, puede acudir a <http://www.php.net>, el sitio oficial de este lenguaje, para encontrar noticias, documentación y manuales sobre el mismo.

```
gotoAndPlay("capitulo_4");
```

Capítulo 4

MySQL

MySQL es uno de los servidores de bases de datos de código abierto (*Open Source*) más populares y conocidos del mundo, un sistema de manejo de bases de datos sin igual en rapidez, estabilidad y facilidad de desarrollo. Dispone, además, de una arquitectura que lo hace extremadamente rápido y fácil de personalizar.

Servidores de bases de datos

Un servidor de bases de datos es un programa que almacena datos estructurados en forma de tablas. El servidor acepta conexiones de clientes a través de un puerto TCP/IP y admite consultas realizadas en lenguaje SQL (*Structured Query Language*) para posteriormente devolver los datos resultantes del procesamiento de aquellas al cliente a través de la Red.

Organización de MySQL

MySQL está conformada por distintas bases de datos, cada una de las cuales consta de una o varias tablas, que son las que contienen la información y que constan de tres elementos principales:

- Columnas.
- Datos.
- Índices.

Columnas

Como las tablas están constituidas por columnas, cada una de estas últimas ha de tener un nombre único, para que podamos referirnos a cada una de ellas sin riesgo de confundirla con alguna de las otras.

Al ser la tabla formada por columnas, cada columna debe tener un nombre único, para así poder referirnos a cada una de ellas específicamente. En MySQL existen tres tipos distintos de columnas, dependiendo de los datos que vayan a almacenar:

- Tipo Numérico.
- Tipo Fecha.
- Tipo Cadena.

Tipo Numérico

Los tipos de datos numéricos pueden dividirse en dos grandes grupos: aquellos que tienen decimales (coma flotante) y aquellos que no los tienen.

Tabla 4.1.

Tipos de datos numéricos disponibles para las tablas MySQL.

Tipo Numérico	Descripción
<i>TinyInt</i>	Número entero con o sin signo. Con signo, el rango de valores válidos se encuentra entre -128 y 127. Sin signo, el rango de valores válidos se encuentra entre 0 a 255.
<i>Bit</i>	Número entero cuyo valor puede ser 0 ó 1.
<i>Bool</i>	Número entero cuyo valor puede ser 0 ó 1.

Tipo Numérico	Descripción
<i>SmallInt</i>	Número entero con o sin signo. Con signo, el rango de valores se encuentra entre -32768 y 32767. Sin signo, el rango de valores se encuentra entre 0 y 65535.
<i>MediumInt</i>	Número entero con o sin signo. Con signo, el rango de valores válidos se encuentra entre -8.388.608 y 8.388.607. Sin signo, el rango de valores válidos se encuentra entre 0 y 6777215.
<i>Integer</i>	Número entero con o sin signo. Con signo, el rango de valores válidos se encuentra entre -2147483648 y 2147483647. Sin signo, el rango de valores válidos se encuentra entre 0 y 429.4967.295.
<i>Int</i>	Número entero con o sin signo. Con signo, el rango de valores válidos se encuentra entre -2147483648 y 2147483647. Sin signo, el rango de valores válidos se encuentra entre 0 y 429.4967.295.

Tipo Numérico	Descripción
<i>BigInt</i>	Número entero con o sin signo. Con signo, el rango de valores válidos se encuentra entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807. Sin signo, el rango de valores válidos se encuentra entre 0 y 18.446.744.073.709.551.615.
<i>Float</i>	Número pequeño en coma flotante de precisión simple. El rango de valores válidos se encuentra entre -3.402823466E+38 y -1.175494351E-38, 0 y entre 1.175494351E-38 y 3.402823466E+38.
<i>xReal</i>	Número en coma flotante de precisión doble. El rango de valores válidos se encuentra entre -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y entre 2.2250738585072014E-308 y 1.7976931348623157E+308.

Tipo Numérico	Descripción
<i>Double</i>	Número en coma flotante de precisión doble. El rango de valores válidos se encuentra entre -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y entre 2.2250738585072014E-308 y 1.7976931348623157E+308.
<i>Decimal</i>	Número en coma flotante desempquetado. En este caso el número se almacena como una cadena.
<i>Dec</i>	Número en coma flotante desempquetado. En este caso el número se almacena como una cadena.
<i>Numeric</i>	Número en coma flotante desempquetado. En este caso el número se almacena como una cadena.



Nota: Expresar un número de la forma 3.402823466E+38 es equivalente a expresarlo de la forma $3.402823466 \times 10^{38}$.

Tipo Fecha

Cuando almacenamos fechas en nuestras tablas, hay que tener en cuenta el hecho de que MySQL no comprueba de forma concreta si una fecha es o no válida, sino si el mes indicado está comprendido entre 0 y 12 y el día del mes está comprendido entre 0 y 31. Atendiendo a este factor, disponemos de varios tipos que podemos utilizar para manejar fechas:

Tabla 4.2.
Tipos de fecha disponibles para las tablas MySQL.

Tipo Fecha	Descripción
Time	Almacena una hora. El rango de valores válidos para la misma se encuentra entre -838 horas, 59 minutos y 59 segundos y 838 horas, 59 minutos y 59 segundos. El formato de almacenamiento es 'HH:MM:SS'.
Year	Almacena un año. El rango de valores válidos para el mismo se encuentra entre el año 1901 y el año 2155. El campo puede tener tamaño 2 o tamaño 4, dependiendo de si deseamos almacenar el año con dos o cuatro dígitos.

Tipo Fecha	Descripción
Date	Almacena una fecha. El rango de valores válidos de la misma se encuentra entre el 1 de enero de 1001 y el 31 de diciembre de 9999. El formato de almacenamiento es <i>año-mes-día</i> .
DateTime	Combinación de fecha y hora. El rango de valores válidos se encuentra entre las 0 horas, 0 minutos y 0 segundos del 1 de enero de 1001 y las 23 horas, 59 minutos y 59 segundos del 31 de diciembre del 9999. El formato de almacenamiento es de <i>año-mes-día horas:minutos:segundos</i> .
TimeStamp	Combinación de fecha y hora. El rango de valores válidos de la misma se encuentra entre el 1 de enero de 1970 y el 31 de Diciembre de 2037. El formato de almacenamiento depende del tamaño del campo como puede observar en la tabla 4.3.

Tabla 4.3.

Formatos de almacenamiento del tipo de fecha *TimeStamp*.

Tamaño	Formato
14	AñoMesDíaHoraMinutoSegundo aaaammddhhmmss.
12	AñoMesDíaHoraMinutoSegundo aammddhhmmss.
8	AñoMesDía aaaammdd.
6	AñoMesDía aammdd.
4	AñoMes aamm.
2	Año aa.

Tipos Cadena

La principal distinción que se hace entre los tipos de cadena disponibles para las tablas MySQL es la de que las cadenas puedan disponer de longitud fija o variable:

Tabla 4.4.

Tipos de cadena disponibles para las tablas MySQL.

Tipo Cadena	Descripción
<i>Char(n)</i>	Almacena una cadena de longitud fija cuyo número de caracteres puede estar comprendido entre 0 y 255.
<i>VarChar(n)</i>	Almacena una cadena de longitud variable cuyo número de caracteres puede estar comprendido entre 0 y 255.

Pero, además de esta distinción, también pueden distinguirse otros dos subtipos dentro de los tipos de fecha, atendiendo al tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Estos dos subtipos son el tipo *Text* y el tipo *BLOB* (*Binary Large Object*), en los que mientras que el primero se ordena sin tener en cuenta mayúsculas y minúsculas, el segundo se ordena teniéndolas en cuenta. Estos últimos, los tipos pertenecientes al subtipo *BLOB*, se utilizan para almacenar datos binarios, como puedan ser los de los ficheros, por ejemplo.

Tabla 4.5.

Subtipos de tipo BLOB disponibles para las tablas MySQL.

Tipo Cadena	Descripción
<i>TinyText</i>	Columna con una longitud máxima de 255 caracteres.
<i>TinyBlob</i>	Columna con una longitud máxima de 255 caracteres.
<i>Blob</i>	Texto con una longitud máxima de 65535 caracteres.
<i>Text</i>	Texto con una longitud máxima de 65535 caracteres.
<i>MediumBlob</i>	Texto con una longitud máxima de 16.777.215 caracteres.
<i>MediumText</i>	Texto con una longitud máxima de 16.777.215 caracteres.
<i>LongBlob</i>	Texto con una longitud máxima de 4.294.967.295 caracteres. Hay que tener cuenta que, debido a los protocolos de comunicación, los paquetes que se envían a través de la Red pueden tener un máximo de 16 Mb.

Tipo Cadena	Descripción
<i>LongText</i>	Texto con una longitud máxima de 4.294.967.295 caracteres. Hay que tener cuenta que, debido a los protocolos de comunicación, los paquetes que se envían a través de la Red pueden tener un máximo de 16 Mb.
<i>Enum</i>	Campo que puede obtener un único valor de una lista de posibles valores donde se especifica. Este tipo acepta hasta 65535 valores distintos.
<i>Set</i>	Campo que puede contener ninguno, uno, o bien varios valores de una lista que puede tener un máximo de 64 valores.

Las columnas, además del tipo al que pertenecen, también pueden tener propiedades o atributos:

- **Longitud:** Número de caracteres que acepta la columna para los valores que se introducen en ella.
- **Default:** Valor predeterminado, es decir, el valor que se asigna a la columna cuando no se le asigna un valor específico.
- **Null - Not Null:** Aceptación o no de valores nulos en caso de que la columna pueda recibir valores nulos.

- **Auto_increment:** Autoincremental, aumenta en una unidad su valor por cada nuevo registro que se añada a la columna.

Datos

Los datos son la información que se almacena por filas dentro de las distintas tablas MySQL de las que disponemos, y cada uno de los registros es del tipo de datos de la columna a la que pertenece.

Índices

Las tablas de MySQL pueden utilizar índices, que sirven para mejorar el tiempo de respuesta de MySQL en las consultas cuando se realizan búsquedas en las distintas tablas. Puede establecer distintos tipos de índices para sus tablas:

Tabla 4.6.

Tipos de índice para las tablas MySQL.

Tipos de Índice	Descripción
<i>Primary</i>	Establece que el campo en el que se define sea primario.
<i>Index</i>	Crea un indexado del campo para que la búsqueda sea más rápida.
<i>Unique</i>	Establece que los valores de ese campo sean únicos para cada nuevos datos de la tabla.



Nota: Es altamente recomendable que cada una de las tablas que cree posea un campo con valores únicos, lo que le ayudará a la identificación de ese registro. Por lo general, este registro único es un número cuyo valor aumenta en una unidad con cada registro nuevo que se añade a la tabla (un número autoincremental), y se le suele dar un nombre que sea identificativo de lo que representa, por ejemplo "id".

Manipulación y utilización de MySQL

Para manipular MySQL puede usted utilizar la terminal o consola del ordenador (una ventana del sistema MS-DOS), pero, por motivos de seguridad, la mayoría de los proveedores de hospedaje de sitios Web no permiten acceder a la existente en el servidor, por lo que suele ofrecerse a cambio el tener instalado un administrador o manipulador para la base de datos.

Para acceder al administrador de bases de datos, abra una nueva ventana de su navegador y teclee:

`http://localhost/phpMyAdmin`

tras lo que deberá ver en su navegador el administrador de bases de datos dispuesto para ser utilizado tal y como se muestra en la figura 4.1.



Nota: Recuerde que para acceder a contenidos alojados en `http://localhost` ha de haber inicializado previamente el servidor Apache.

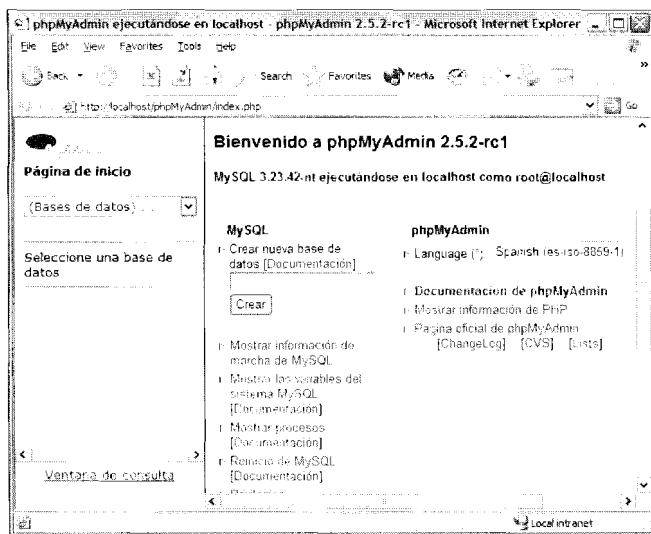


Figura 4.1.

Pantalla de bienvenida al administrador de bases de datos.

Para irnos acostumbrando al funcionamiento de MySQL, vamos a crear primeramente una base de datos, y posteriormente, y dentro de ésta, crearemos una tabla sobre la que haremos inserciones, modificaciones y selecciones de datos, de modo que las estructuras a crear en los siguientes apartados le servirán además para ir realizando las distintas prácticas y ejercicios necesarios para familiarizarse con la manipulación de las bases de datos instaladas en su servidor.

Creación de una base de datos

Para comenzar, vamos a crear una nueva base de datos llamada *pruebas* en su administrador

de bases de datos (*phpMyAdmin*). Para ello, teclee:

```
pruebas
```

en el cuadro de introducción de texto que se encuentra debajo de **Crear nueva base de datos** y pulse el botón **Crear**.

Si desea utilizar la consola (para lo puede presionar el enlace **Ventana de consulta** del administrador) para crear la base de datos, teclee en la misma:

```
CREATE DATABASE pruebas
```

Creación de una tabla para la base de datos

Una vez que hemos creado la base de datos, necesitaremos al menos una tabla en la que almacenar información que posteriormente manipularemos. Para ello, vamos a crear una tabla de nombre *directorio* y cuya estructura puede observar en la tabla 4.7.

Tabla 4.7.

Estructura de la tabla *directorio* de la base de datos *pruebas*.

Nombre del campo	Tipo de datos
id	Entero mediano, autoincremental.
nombre	Varchar de 20 caract.
apellido	Varchar de 20 caract.
email	Varchar de 30 caract.
url	Varchar de 30 caract.
nick	Varchar de 20 caract.

El administrador de la base de datos nos ofrece dos posibilidades para crear una nueva tabla. La primera de ellas consiste en utilizar un formulario escrito en HTML, para lo cual hemos de comenzar seleccionando en el menú de bases de datos que se encuentra a la izquierda de su administrador la base de datos en la que vamos a crear la tabla; *pruebas* en nuestro caso.

Una vez seleccionada la base de datos *pruebas* accederá a una nueva pantalla en la que podrá observar dos cuadros de introducción de texto debajo de **Crear nueva tabla en la base de datos pruebas**, siendo el primero de ellos para el nombre de la nueva tabla y el segundo para el número de campos de que constará ésta última, como puede ver en la figura 4.2. Tecleamos:

directorio

en el primer cuadro de introducción de texto y:

6

en el segundo, para posteriormente pulsar el botón **Continúe**.

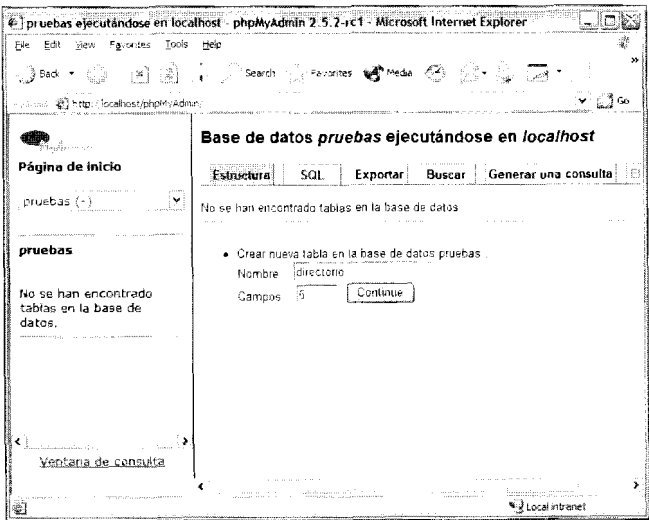


Figura 4.2.

Creación de la tabla *directorio* usando *phpMyAdmin*.

Veremos entonces en nuestro navegador una serie de campos desplegables y de introducción de texto que deberemos rellenar de acuerdo a la estructura que decidimos que había de tener la tabla *directorio* y que podemos observar en la tabla 4.7. Una vez completados los campos del formulario (como se puede ver en la figura 4.3), pulsamos el botón **Grabar**, con lo que se creará la tabla *directorio* en MySQL, y veremos un mensaje que nos informa del éxito de la operación.

Esta forma de crear las distintas tablas de nuestra base de datos es sencilla de utilizar, pero tiene el inconveniente de que hemos de crear las tablas una por una y, además, nos obliga a rellenar tantos campos como le indi-

quemos al crear cada una de ellas, con lo que si éstas son grandes o tenemos un elevado número de tablas, su creación puede ser una tarea bastante incómoda. Por ello, MySQL ofrece la posibilidad de importar datos externos, es decir, leer un archivo con la información y estructura (e incluso con los datos) de una o varias tablas, de forma que podemos teclear los comandos que necesitamos en un documento de extensión .sql generado con un editor de texto y posteriormente importarlo desde MySQL para que se ejecuten dichos comandos y se creen las tablas con sus respectivas estructuras.

Como ejemplo práctico de lo que acabamos de leer, vamos a seleccionar la base de datos

Base de datos *pruebas* - Tabla *directorio* ejecutándose en *localhost*

Campo	Tipo [Documentación]	Longitud/Valores*	Atributos	Nulo	Predeterminado**	Extra	Primaria	Índice Único	...	Texto complet
id	MEDIUMINT		UNSIGNED	not null		auto_increment	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
nombre	VARCHAR	20		not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
apellidos	VARCHAR	20		not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
email	VARCHAR	30		not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
url	VARCHAR	30		not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
nick	VARCHAR	20		not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>

Comentarios de la tabla:

Tipo de tabla: Predeterminado

* Si el tipo de campo es "enum" o "set" por favor ingrese los valores usando este formato: 'a' 'b' 'c'

Figura 4.3.
Estructura de la tabla directorio.

pruebas en el menú de la izquierda del administrador, y posteriormente vamos a pulsar en la pestaña que lleva el texto SQL en la parte derecha, lo que nos lleva a una pantalla con un área de texto y un campo de texto de tipo Archivo junto a un botón **Browse (Examinar)**. Pulsaremos sobre éste para buscar y seleccionar el documento `material/cap4/sql/`

`directorio_estructura.sql` del CD, como puede verse en la figura 4.4.

Este documento .sql contiene, como puede ver en la figura 4.5, los comandos necesarios para crear la tabla *directorio* en la base de *datos pruebas*.

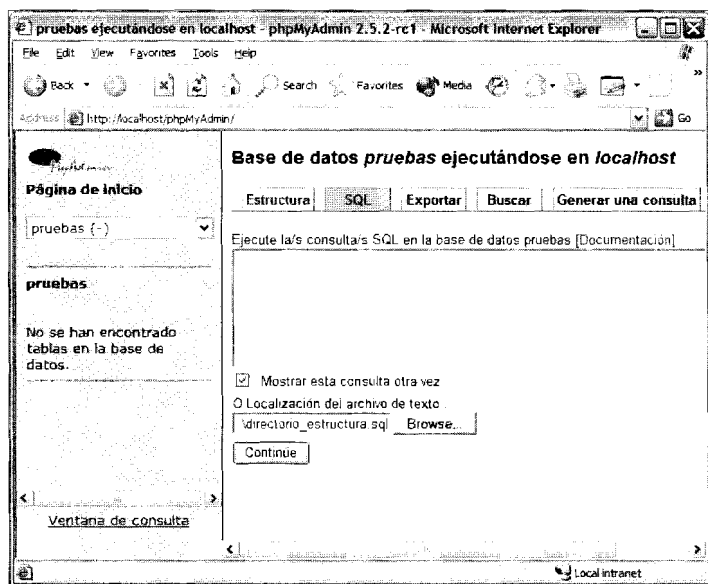


Figura 4.4.
Carga de documentos .sql externos.

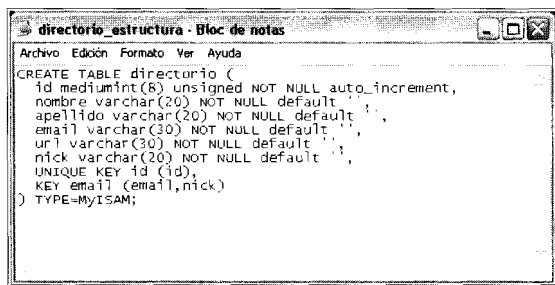


Figura 4.5.
Contenidos del documento
`directorio_estructura.sql`.

Seleccionado el documento, presionamos el botón **Continúe**, y veremos una pantalla que nos informa del éxito de la creación de la tabla (figura 4.6).

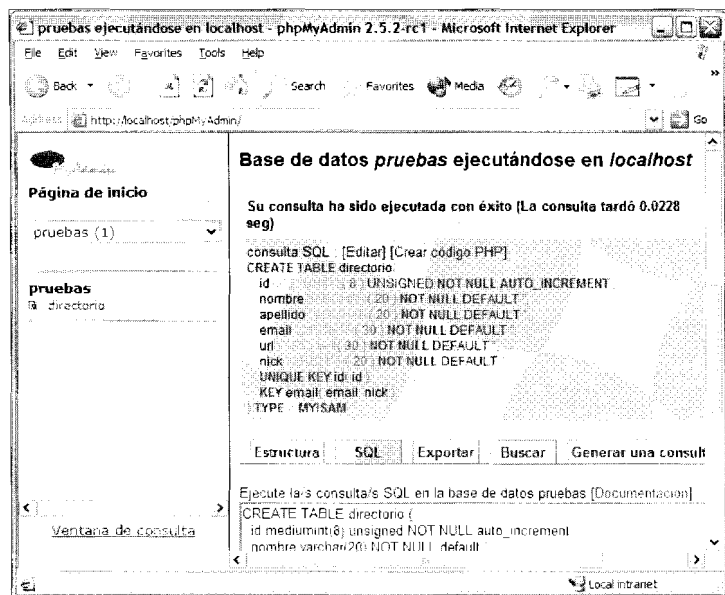


Figura 4.6.

Creación exitosa de la tabla directorio.

Este último método es muy útil, no sólo para ganar tiempo a la hora de crear varias tablas, sino también para exportar la estructura de las bases de datos cuando se instalan módulos pre-programados.

Inserción de información

Antes de insertar datos en la tabla, vamos a importar un archivo con datos de ejemplo con el fin de que, a la hora de realizar selecciones de registros en ejercicios posteriores, las informaciones sean las mismas. Al igual que hicimos con anterioridad para crear una tabla utilizando un documento .sql externo, seleccionaremos la base de datos *pruebas* en el menú de la parte izquierda del administrador y luego la tabla *directorio* del menú de tablas

dentro de la base de datos *pruebas*. A continuación, pulsamos sobre la pestaña de nombre **SQL** de la parte derecha. Utilizando el botón **Browse (Examinar)** buscamos y seleccionamos el documento *material/cap4/sql/directorio_datos.sql* del CD y luego pulsamos el botón **Continúe** para que se realicen las acciones que hemos tecleado en nuestro documento .sql externo.

Una vez almacenada en la tabla *directorio* la información del archivo importado, procederemos a insertar información. Para ello, podemos pulsar sobre el vínculo que reza **Insertar**, lo cual nos permitirá introducir registros nuevos a la tabla de uno en uno, con lo que esto puede conllevar de pérdida de tiempo. Para evitar este problema, vamos a introducir la información mediante sentencias SQL. Con

la tabla *directorio* seleccionada en la parte izquierda del administrador, pulse sobre la pestaña SQL de la parte derecha, en la que puede ver un área de texto en el que vamos a introducir nuestros comandos.

Para insertar información utilice la palabra reservada `INSERT` seguida del nombre de la tabla donde se realizará la inserción. Seguidamente, especificamos los campos que serán afectados cuando realizamos una nueva inserción de un registro y, por último, especificamos los valores que tendrán cada uno de esos campos. Resumiendo, la sintaxis debe ser como sigue:

```
INSERT INTO [nombre_tabla] (campo1,
campo2, campo3) VALUES
('valor_campo1', 'valor_campo2',
'valor_campo3');
```

Si recuerda la estructura de la tabla *directorio* (tabla 4.7), ésta cuenta con los campos *id*, *nombre*, *apellido*, *email*, *url* y *nick*; hemos de tener en cuenta que *id* es autoincremental, con lo que no es necesario asignarle información, puesto que MySQL se encargará de insertar el registro con el número correspondiente. Para comprobar lo que acabamos de decir, vamos a insertar un nuevo registro:

Tabla 4.8.
Datos del nuevo registro.

Campo	Contenido
nombre	Ricardo
apellido	Salazar
email	info@nomaster.com
url	http://www.nomaster.com
nick	Dasso

Teclee en el área de texto la consulta para insertar el registro tal y como hemos especificado con anterioridad:

```
INSERT INTO directorio
(nombre, apellido, email, url, nick)
VALUES
('Ricardo', 'Salazar', 'info@nomaster.com',
'http://www.nomaster.com', 'dasso');
```



Nota: Tenga en cuenta que los valores para cada uno de los campos se teclean entre comillas simples, no así los nombres de los campos.

Pulsando el botón **Insertar** se agregará el nuevo registro a la tabla *directorio* de la base de datos *pruebas*, como puede ver en la figura 4.7.

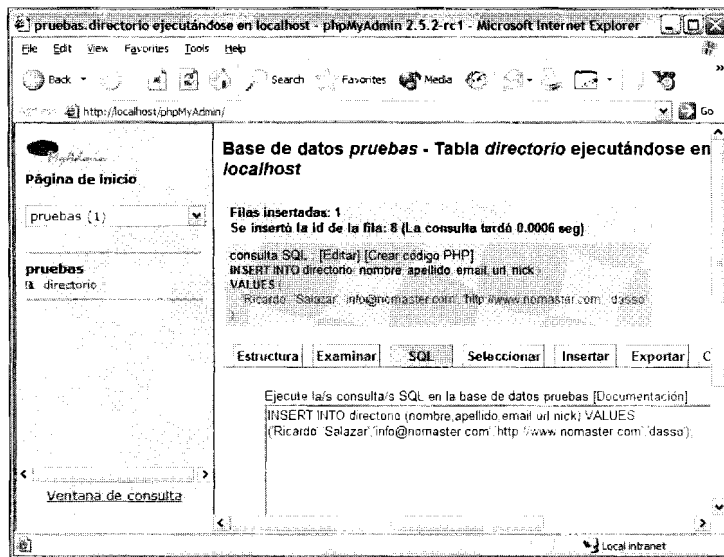


Figura 4.7.
Datos insertados con éxito.

De esta forma hemos realizado la inserción de un solo registro. Si deseamos insertar, por ejemplo, quince, no tenemos más que escribir una sentencia `INSERT INTO...` por cada uno de los registros a añadir en un documento `.sql` y, posteriormente, importarlo a MySQL, tal y como hemos realizado con los datos del documento `directorio_datos.sql` que hemos visto al inicio de este apartado.

Selección de registros de la tabla

La selección de información es una de las actividades más realizadas cuando trabajamos con bases de datos. Cuando realizamos una búsqueda o desplegamos una serie de registros estamos realizando una selección. La

sintaxis para realizar una selección de registros de una tabla es variable, siendo la forma básica como se indica a continuación:

```
SELECT * FROM [nombre_tabla];
```

En este caso, el carácter `*` representa todas las columnas y todos los registros de la tabla.

Vamos a realizar diversas consultas de selección con los datos de nuestra tabla *directorio*. Seleccionamos dicha tabla del menú de tablas de la base de datos *pruebas* en la parte izquierda del administrador, y a continuación pulsamos sobre la pestaña **SQL**. En el área de texto teclee lo siguiente:

```
SELECT * FROM directorio;
```

La siguiente pantalla mostrará todos los registros de la tabla, como puede observar en la figura 4.8.

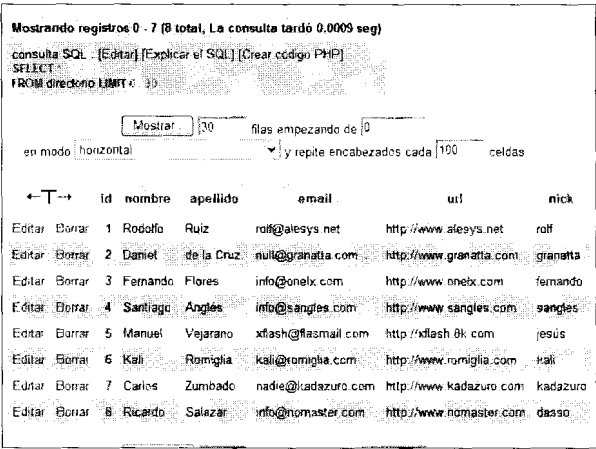


Figura 4.8.
Selección de todos los registros.

En cambio, si deseamos seleccionar sólo algunos campos de la tabla, habremos de separar los nombres de las columnas por comas, como se indica a continuación:

```
SELECT nombre_columna_1,  
nombre_columna_2, ...,  
nombre_columna_N FROM  
[nombre_tabla];
```

De esta forma, si deseamos seleccionar únicamente las columnas *nombre*, *apellido*, *url* y *nick*

de la tabla *directorio* habremos de teclear en el área de texto de SQL lo siguiente:

```
SELECT nombre,apellido,url,nick FROM  
directorio;
```

La siguiente pantalla mostrará únicamente los campos *nombre*, *apellido*, *url* y *nick* de cada registro almacenado en la tabla, como puede observar en la figura 4.9.

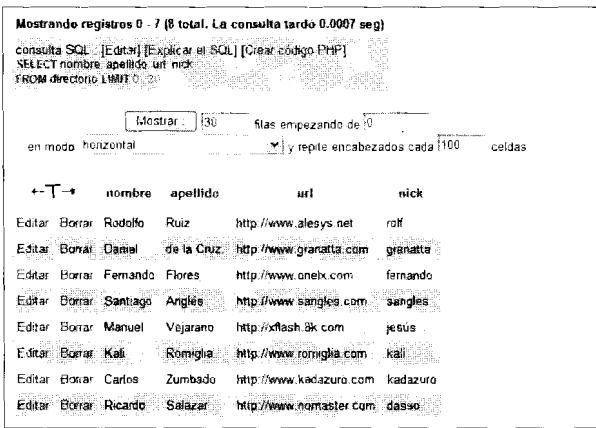


Figura 4.9.
Selección de campos concretos de los registros.

Ordenación de los resultados

Además de obtener los campos deseados en la consulta, también podemos ordenarlos en orden ascendente (ASC) o descendente (DESC), referenciando a una columna o a varias, para lo cual agregaremos ORDER BY [columna] [modo de ordenamiento] a la consulta, resultando entonces lo siguiente:

```
SELECT nombre_columna_1,
nombre_columna_2, ...,
nombre_columna_N FROM [nombre_tabla]
ORDER BY [nombre_columna] [modo de
ordenamiento];
```

Si queremos seleccionar registros en nuestra tabla *directorio* tomando como referencia al campo *nombre*, y en orden descendente, teclearíamos:

```
SELECT nombre, apellido, url, nick FROM
directorio ORDER BY nombre DESC;
```

con lo que, como puede verse en la figura 4.10, se muestran los distintos registros pero ordenados no por el campo *id*, sino por el campo *nombre*. Al haber indicado además que la ordenación fuese descendente, aparecen en los primeros lugares aquellos nombres cuya inicial es posterior en orden alfabético.

Podemos además ordenar los resultados en forma ascendente ASC o descendente DESC, en base a una columna o varios; para esto agregamos ORDER BY [columna] [modo de ordenamiento] a la consulta. Por ejemplo, para seleccionar registros en base al campo *nombre* en forma descendente teclearíamos:

```
SELECT nombre, apellido, url, nick FROM
directorio ORDER BY nombre DESC;
```

Mostrando registros 0 - 7 (8 total). La consulta tardó 0.0253 seg)

consulta SQL [Editar] [Explicar el SQL] [Crear código PHP]

SELECT nombre, apellido, url, nick
FROM directorio
ORDER BY nombre DESC LIMIT 30

Mostrar: 30 filas empezando de 0

en modo: horizontal y repite encabezados cada: 100 celdas

←T→

	nombre	apellido	url	nick
Editar Borrar	Santiago	Anglés	http://www.sangles.com	sangles
Editar Borrar	Rodolfo	Ruiz	http://www.aleysys.net	roif
Editar Borrar	Ricardo	Salazar	http://www.nomaster.com	dasso
Editar Borrar	Manuel	Vejarano	http://xfish.5k.com	jesús
Editar Borrar	Kali	Romiglia	http://www.romiglia.com	kali
Editar Borrar	Fernando	Flores	http://www.oneix.com	fernando
Editar Borrar	Daniel	de la Cruz	http://www.granatta.com	granatta
Editar Borrar	Carlos	Zumbado	http://www.kadazuro.com	kadazuro

Figura 4.10.

Selección y ordenamiento de resultados.

Puede practicar las selecciones ordenando por distintas columnas. Para ordenar por más de una columna, puede adjuntar el nombre del campo y el orden, seguido cada uno de ellos por la instrucción ORDER BY, por ejemplo:

```
SELECT nombre, apellido, url, nick
FROM directorio ORDER BY nombre ASC,
url DESC;
```

Selección de datos específicos

En caso de que deseemos realizar una consulta a la base de datos buscando los registros que concuerden con un criterio específico, utilizaremos la cláusula WHERE, que se suele utilizar de la siguiente forma:

```
SELECT [nombre_columnas] FROM
[nombre_tabla] WHERE
[nombre_columna] [criterio_comparación] [valor];
```

donde *nombre_columna* es una columna de la tabla, *criterio_comparación* es el lugar en el que indicamos a la base de datos que seleccione los campos donde la informa-

ción sea "igual", "mayor" o "menor" o que contenga cierto valor, y *valor* es el criterio de búsqueda de la selección que estamos llevando a cabo.

Por ejemplo, si quisiéramos seleccionar todas las columnas de la tabla *directorio* donde el nombre sea *Ricardo*, teclearíamos lo siguiente:

```
SELECT * FROM directorio WHERE
nombre = 'Ricardo';
```

Si deseamos seleccionar todas las columnas de la tabla *directorio* donde *nombre* comienza con la letra *R*, teclearíamos lo siguiente:

```
SELECT * FROM directorio WHERE
nombre LIKE 'R%';
```

Si deseamos seleccionar todas las columnas de la tabla *directorio* donde *nombre* contenga la letra *r*, teclearíamos lo siguiente:

```
SELECT * FROM directorio WHERE
nombre LIKE '%R%';
```

Observe los registros de la tabla *directorio* y practique realizando distintos tipos de consultas de selección como las anteriores.

Mostrar un determinado número de registros

Puede que cuando realizamos una consulta no queramos mostrar todos los registros resultantes en la búsqueda, sino simplemente un número determinado de ellos. Para este tipo de consultas se utilizan lo que se denominan *límites*, y la sintaxis para usarlos es la siguiente:

```
SELECT [campos] FROM [tabla] [WHERE]
LIMIT[inicio],[cantidad];
```

siendo *inicio* el índice en el *array* resultante de respuestas y *cantidad* el número de registros a mostrar.

Por ejemplo, si queremos mostrar solamente los dos primeros registros de la tabla *directorio*, teclearíamos:

```
SELECT * FROM directorio LIMIT 0,2;
```

Si queremos mostrar los dos registros siguientes, la consulta a realizar sería:

```
SELECT * FROM directorio LIMIT 2,2;
```



Nota: Recuerde que puede pensar en un *array* como en uno de esos grandes ficheros con cajones, dentro de cada uno de los cuales hay información guardada sobre un tema concreto. Los *arrays* son estructuras que sirven para almacenar más de un valor al mismo tiempo, en distintas celdas, y disponen de un índice, un número identificador que se usa para acceder a cada uno de los datos almacenados. Este índice es un número que va desde 0 hasta *número_valores_del_array-1*; así, si un *array* guarda seis valores, el índice de dicho *array* va desde 0 hasta 5. Por ello, los dos primeros elementos de un *array* están almacenados en las posiciones 0 y 1, de forma que cuando queremos escoger los dos primeros registros de la consulta comenzamos por el número 0.

Obsérvese como se siguen mostrando dos registros tras la búsqueda, pero al variar el valor de *inicio*, el lugar a partir del cual se empiezan a mostrar registros también varía. Este tipo de consultas (utilizando límites) es muy útil cuando deseamos realizar paginaciones de resultados.

Modificación de registros almacenados en la tabla

Para actualizar o modificar el contenido de alguno de los registros almacenados en la tabla se utiliza como referencia el campo cuyo identificador es único para cada registro. En nuestra tabla *directorio*, el campo *id* es único por ser autoincremental, por lo que nos ayuda a especificar cuál es el registro que deseamos modificar.

La sintaxis para realizar modificaciones es la siguiente:

```
UPDATE [nombre_tabla] SET nombre_campo1=valor1, nombre_campo2=valor2, ...,
nombre_campoN=valorN [WHERE][nombre_columna][criterio_comparación][valor];
```

Inserte el siguiente registro en la tabla *directorio* tal y como hicimos con anterioridad:

```
INSERT INTO directorio (nombre,apellido,email,url,nick) VALUES
('Juan','Perez','info@juanperez.com','juanperez.com','jperez');
```

A continuación seleccionaremos el registro que acabamos de insertar. Esto podemos hacerlo de varias formas, seleccionando en la tabla al usuario cuyo nombre es *Juan*, por ejemplo:

```
SELECT * FROM directorio WHERE nombre= 'Juan';
```

También podemos seleccionar de la tabla *directorio* el registro en el que el valor del campo *id* sea el mayor (al ser este campo autoincremental el último registro será que el tenga mayor *id*), como se muestra en la figura 4.11.

```
SELECT * FROM directorio ORDER BY id
DESC LIMIT 0,1;
```

Si se ha fijado, el nuevo registro que hemos añadido a la tabla tiene la particularidad de que algunos de los contenidos de sus campos están incompletos (en comparación con los datos almacenados de los demás registros); el apellido de *Juan* carece de tilde, y su *url* no contiene la cadena `http://www.` que sí tienen el resto de registros, por lo que hemos de modificar los datos que hemos almacenado previamente (concretamente los campos *apellido* y *url*) utilizando el campo *id* como identificador único.

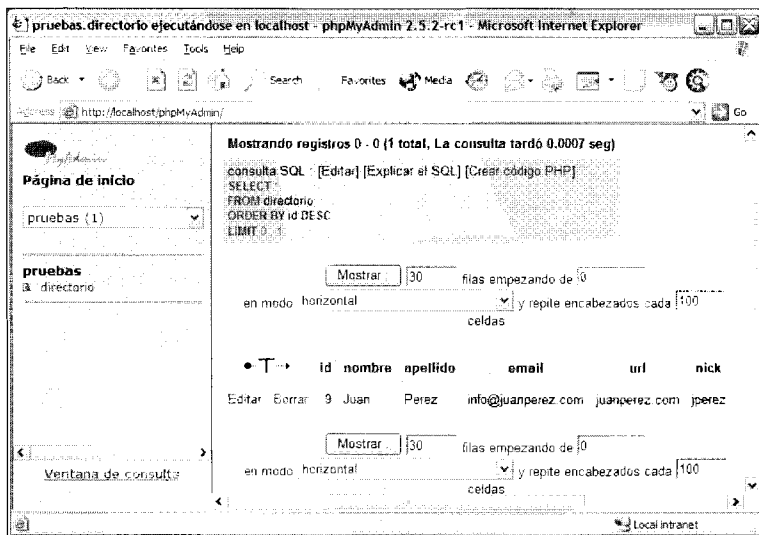


Figura 4.11.

Selección del último registro insertado.

Para ello teclearemos:

```
UPDATE directorio SET
apellido='Pérez',url='http://
www.juanperez.com' WHERE id=9;
```

valor del campo *id* el del último registro insertado en la tabla:

```
SELECT * FROM directorio ORDER BY id
DESC LIMIT 0,1;
```

Ejecute la consulta a la base de datos, y seleccione de nuevo el registro con utilizando como

Como puede observar en la figura 4.12, los datos se han modificado con éxito, con lo que el último registro añadido ya dispone de la información correcta.

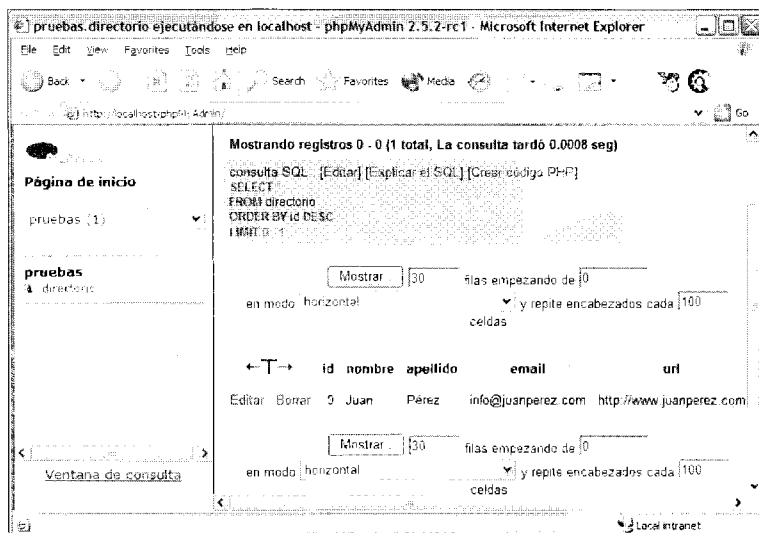


Figura 4.12.

Modificación del último registro añadido a la tabla directorio.

El utilizar un identificador único para realizar las modificaciones es fundamental. En el supuesto de que omitiera usar dicho identificador (*id* en este ejemplo), la consulta a la base de datos se haría de la siguiente forma:

```
UPDATE directorio SET  
apellido='Pérez',url='http://  
www.juanperez.com';
```

Y como resultado de esta acción, todos los registros almacenarían en su campo *apellido* el valor *Pérez* y en su campo *url* el valor *http://www.juanperez.com*.

Eliminación de registros almacenados en la tabla

Para realizar el borrado de un registro se utiliza la siguiente sintaxis:

```
DELETE FROM [tabla] WHERE [cláusula  
where];
```

Para borrar el registro cuyo identificador *id* es 9, la consulta a efectuar sería:

```
DELETE FROM directorio WHERE id=9;
```



Nota: Es muy importante hacer referencia a una columna con un identificador único a la hora de realizar la consulta. De igual forma a como ocurría en una consulta de actualización, si se omite la cláusula *where* al efectuar una operación de borrado, la consulta afecta a todos los registros, por lo que en este caso serían eliminados todos ellos.

Más información sobre MySQL

Para más información sobre MySQL puede acudir a <http://www.mysql.com>, el sitio oficial de este software, para encontrar noticias, documentación y descargar las nuevas versiones del mismo.

```
gotoAndPlay("capitulo_5");
```

Capítulo 5

MySQL y PHP

En los dos capítulos anteriores hemos aprendido un uso básico de MySQL, así como algunas características del lenguaje PHP. Ahora aprenderemos a utilizar ambos de forma conjunta, realizando los mismos ejercicios que en el capítulo anterior pero utilizando PHP para comunicarnos con la base de datos y mostrar los resultados en la ventana de su navegador.

Al igual en los capítulos anteriores, puede encontrar los documentos a cuyo código hacemos referencia en este capítulo en la carpeta `material/cap5/php/` del CD, por si en un determinado momento sus ejercicios no funcionaran o no lo hicieran correctamente.

MySQL y PHP

PHP dispone de funciones predefinidas específicas para comunicarse de forma efectiva con MySQL mediante un proceso que por lo general suele desarrollarse pasando por las siguientes etapas:

- Conexión de PHP a la base de datos MySQL.
- Realización de una consulta a la base de datos MySQL (SQL *query*).
- Impresión en pantalla o procesamiento de los datos obtenidos.
- Liberación de resultados y cierre de la conexión con la base de datos MySQL.

MySQL cierra automáticamente las conexiones con la base de datos una vez procesado el documento .php que la abrió, pero para estar más seguros debemos siempre cerrar todas las conexiones que establezcamos.

Ficheros de configuración y acciones repetitivas

Cada vez que deseemos comunicarnos con MySQL a través de documentos escritos en PHP hay que establecer una conexión en la que necesitaremos cuatro variables, las cuales, por lo general, se utilizarán en todos los documentos PHP del sitio o proyecto que se esté realizando. Para utilizar estas variables disponemos de dos posibilidades:

- Definir las variables en cada documento PHP.
- Definir las variables en un fichero externo que es incluido para su uso en cada documento PHP.

Ambas opciones funcionan correctamente, y el uso es el mismo, pero a la hora de hacer modificaciones (en el nombre de la base de datos, la contraseña o el nombre de usuario) será más fácil y efectivo realizarlas una sola vez sobre el fichero externo que realizar la modificación una vez por cada documento PHP que vaya a utilizar.

Por otra parte, cada vez que necesite conectarse desde un documento .php a la base de datos MySQL, realizará el mismo proceso una y otra vez; de hecho, al transcurrir el tiempo, nos percataremos de que utilizamos una y otra vez las mismas funciones, por lo que es

recomendable guardarlas en archivos externos de forma que nuestros proyectos dispongan de una mejor organización y sean más fáciles de actualizar o corregir en caso de que existan fallos (es un buen método para centralizar los posibles errores; puesto que agrupamos las funciones en un fichero concreto; en caso de producirse un fallo sólo habremos de buscar el error en dicho fichero y, cuando sea subsanado, todos los documentos .php que hacen uso de este fichero externo funcionarán correctamente).

Vamos a crear ahora en la raíz del servidor local una carpeta de nombre `phpmysql`, y dentro de ésta crearemos otra de nombre `includes`, en la que vamos a almacenar todos los archivos de uso común y de configuración.

Obtener siempre una respuesta

Es importante y conveniente que el documento .php siempre imprima algo, incluso en el caso de que no se haya encontrado resultado alguno o no haya podido establecerse conexión con la base de datos. Para tal efecto, podemos utilizar la función `die`, siendo su sintaxis como sigue:

```
die("mensaje");
```

De esta forma, cuando se realice una consulta o conexión a la base de datos, podemos establecer el uso de esta función como alternativa en caso de error. Por ejemplo:

```
$resultado = mysql_connect("servidor","usuario","clave") or die("No se pudo conectar al servidor");  
$resultado = mysql_query ("SELECT * FROM tabla") or die("No se pudo conectar al servidor");
```

Si se produce un error, ambos ejemplos mostrarán el texto "No se pudo conectar al servidor". Si lo que desea saber es el error exacto que se produjo, puede hacer uso de la función `mysql_error()`, que retornará el error que se produjo:

```
$resultado = mysql_connect("servidor","usuario","clave") or die(mysql_error());  
$resultado = mysql_query ("SELECT * FROM tabla") or die(mysql_error());
```

El uso de la función *die* le indica a *PHP* que termine de interpretar el documento en el que se encuentra después de imprimir el mensaje; es decir, una vez impreso, se terminan los procesos de la página.

Un aspecto a tener en cuenta es el hecho de que la función `mysql_error()` proporciona el error exacto que se produjo en MySQL, lo cual es de gran ayuda cuando se está desarrollando el proyecto. Sin embargo, no es necesario suministrar tanta información al usuario que esté navegando por nuestro sitio, por lo que es conveniente, una vez terminado el trabajo y comprobado que todo funciona correctamente, sustituir nuestros mensajes de error por otros más sencillos en los que no se den muchos detalles.

Uso de MySQL con PHP (identificadores y arrays)

Establecer una conexión con MySQL

Para establecer una conexión con MySQL se requieren el nombre del servidor, el nombre de usuario y una contraseña, información que proporciona el administrador del servicio cuando se contrata un servicio de hospedaje en Web. La función que se utiliza es `mysql_connect`, cuya sintaxis es la siguiente:

```
$cnx = mysql_connect("servidor","usuario","contraseña");
```

Si hemos de especificar que el servidor es accedido por un puerto concreto, hemos de adjuntarlo al nombre del servidor separado por el carácter ":":

```
$cnx = mysql_connect("servidor:3306","usuario","contraseña");
```

El identificador del enlace que se establece se almacena, en este caso, en la variable `cnx`, aunque puede establecer el nombre de variable que usted desee. En caso de no poder conectar con la base de datos es conveniente que tanto PHP como MySQL nos mantengan informados de lo que está ocurriendo, utilizando para ello la función `die` de la que hablamos con anterioridad, cuya sintaxis, mostramos de nuevo:

Por ejemplo, para seleccionar la base de datos de nombre `pruebas` (que utilizamos con anterioridad en el capítulo 4 dedicado a MySQL) utilizando el identificador de conexión almacenado en la variable `$cnx` utilizaríamos la siguiente línea:

```
mysql_select_db("pruebas",$cnx);
```

```
$cnx = mysql_connect("servidor","usuario","contraseña") or die (mysql_error());  
$cnx = mysql_connect("servidor","usuario","contraseña") or die ("No se puedo  
conectar con el servidor");
```

Por ejemplo, si es usted el administrador del sitio local en el que está haciendo las pruebas puede tratar de conectarse al servidor utilizando la siguiente línea de código:

```
$cnx = mysql_connect("localhost","root","root") or die (mysql_error());
```

Selección de la base de datos

Una vez establecida con éxito la conexión con MySQL, hemos de seleccionar la base de datos con la que vamos a trabajar, para lo que utilizaremos la función `mysql_select_db`, cuya sintaxis es la siguiente:

```
mysql_select_db(base de datos, identificador de conexión);
```

Creación de los ficheros de configuración

Por lo general (no siempre) trabajaremos conjuntamente con una base de datos y un servidor de bases de datos, por lo que es conveniente crear archivos de configuración, como ya se comentó anteriormente en este mismo capítulo. Vamos a utilizar dos ficheros, uno de ellos almacenará las variables de configuración y el otro lo utilizaremos para agrupar las funciones de uso común.

En el primero de ellos, que puede encontrar con el nombre `config.php` en la carpeta `material/cap5/php/includes/` del CD, especificaremos los valores de servidor, usuario, contraseña y base de datos en distintas variables, y aunque posteriormente

agreguemos más información en este fichero, de momento el contenido será el siguiente:

```
<?php
$HOSTNAME = "localhost"; //SERVIDOR
$USERNAME = "root";      //USUARIO
$PASSWORD = "root";      //CONTRASEÑA
$DATABASE = "pruebas";   //BASE DE DATOS
?>
```

Guarde este documento con el nombre `config.php` en la carpeta `phpmysql/` `includes` de su servidor local, que creó anteriormente, y abra un nuevo documento de texto para crear el segundo archivo.

Este segundo archivo, y aunque con posterioridad le añadamos más funciones que utilicemos a medida que las necesitemos, va a comenzar conteniendo funciones relativas al hecho de que cada vez que queramos obtener o modificar información de la base de datos vamos a tener que conectarnos con ella, lo que se producirá en todos los documentos `.php` que utilicemos, por lo que vamos a declarar una función que nos sirva para establecer una conexión con la base de datos:

```
<?
/**
función conectar
que = se conecta a MySQL y devuelve el identificador de conexión
***/
function conectar(){
    global $HOSTNAME,$USERNAME,$PASSWORD,$DATABASE;
    $idcnx = mysql_connect($HOSTNAME, $USERNAME, $PASSWORD) or
        DIE(mysql_error());
    mysql_select_db($DATABASE, $idcnx);
    return $idcnx;
}
?>
```

Este archivo, del que puede encontrar una copia de nombre `funciones.php` en la carpeta `material/cap5/php/includes/` del CD, tiene algunas características que pasamos a comentar:

- La función contiene comentarios, lo cual es de utilidad en proyectos grandes, ya que con leer el comentario sabemos para qué sirve la función y no tenemos por qué analizarla para averiguarlo.
- La función toma los valores de las variables del archivo `config.php`, que son globales y externas a la función.

Guarde ahora el documento con el nombre `funciones.php` en la carpeta `phpmysql/includes` de su servidor local. Ahora que dispone de los dos ficheros, y teniendo en cuenta que `funciones.php` necesita a `config.php` para utilizar las variables, es fácil hacerse una idea del orden en el que hay que incluir estos ficheros en las llamadas para que sean utilizados: primero el de configuración y, seguidamente, el de funciones.

Realización de consultas a MySQL

Establecida la conexión con MySQL y seleccionada la base de datos con la que vamos a trabajar podemos realizar las consultas que queramos a MySQL, para lo que usamos la función `mysql_query`, cuya sintaxis es la siguiente:

```
$resultado = mysql_query(Consulta
SQL, identificador de conexión);
```

Si no especificamos un identificador de conexión, MySQL asume que se utiliza el último creado.

Por ejemplo, para realizar una consulta que obtenga todas las filas de la tabla *directorio* y las ordene en orden descendente en base al valor de *id*, utilizaríamos:

```
$res= mysql_query("Select * FROM
directorio ORDER BY id desc") or die
(mysql_error());
```

El resultado de la consulta se almacena en la variable `$res`.

Impresión de resultados

Hay varias formas de procesar los resultados que devuelve MySQL, pero antes de hacerlo siempre es conveniente verificar si se obtuvieron resultados y no se produjo ningún error. Al retornar MySQL el resultado en forma de filas, nos basta con comprobar si dicho número de filas es mayor que cero, para lo que utilizamos la función `mysql_num_rows`, cuya sintaxis es:

```
mysql_num_rows(identificador de
resultado);
```

Puede ver un ejemplo en el documento `material/cap5/php/mysql_num_rows.php` del CD, en el que encontrará el siguiente código:

```

1<?php
2include ("includes/config.php");
3include ("includes/funciones.php");
4
5//nos conectamos a mysql
6$cnx = conectar ();
7//consulta.
8$sql = "SELECT * FROM directorio ORDER BY id ASC";
9$res= mysql_query($sql) or die (mysql_error());
10if (mysql_num_rows($res) >0){
11 echo "cantidad de filas en el resultado: " . mysql_num_rows($res);
12}else{
13 echo "no se obtuvieron resultados";
14}
15mysql_close($cnx);
16?>

```



Nota: Hemos numerado las líneas para poder hacer referencia más rápidamente a las actividades que desarrolla cada fichero .php, pero omítalas a la hora de crear los suyos propios.

En las líneas 2 y 3 se incluyen los ficheros de configuración, y en la 6 se realiza una llamada a la función `conectar` (que se encuentra declarada en el fichero `funciones`, como vimos al crear los ficheros de configuración), asignando entonces el identificador de conexión a una variable de nombre `$cnx`.

En la línea 8 almacenamos la consulta en una variable llamada `$sql`, que utilizamos en la línea 9 para realizar la consulta SQL a MySQL. En caso de que se produzca un fallo, se imprime cuál fue el error concreto ("`or die mysql_error()`").

En las líneas 10-14 encontramos una estructura de condición en la que se revisa mediante una sentencia `if` si el número de columnas obtenido una vez realizada la consulta es

mayor que cero. De ser así, se imprime un mensaje con la cantidad de columnas obtenidas (figura 5.1), y si no hay resultado alguno, se muestra otro mensaje informando de tal circunstancia.

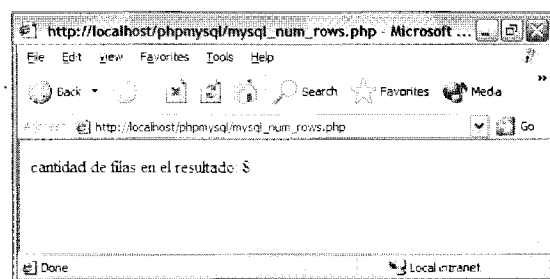


Figura 5.1.

Cantidad de filas obtenidas en el resultado de la consulta SQL.

Finalmente, en la línea 15 cerramos la conexión con la base de datos, utilizando para ello el identificador de conexión.

Podemos proceder de varias maneras si encontramos resultados para imprimir, utilizando para ello las distintas funciones predeterminadas que PHP proporciona para tal efecto. A continuación mostraremos las de

uso más generalizado, aunque el uso de las mismas se hace totalmente al gusto de cada cual.

La extracción de datos suele realizarse utilizando un bucle de tipo `while`, asignando para las filas una variable que es accedida para imprimir los resultados; dependiendo del método que se utilice, la variable podrá disponer de la información de la fila en forma de *array* (*array* de lista u objeto), siendo en este aspecto donde radica la mayor diferencia entre las distintas formas de imprimir resultados. Como ejemplo, obtendremos los contenidos las columnas `nick`, `email` y `url` de la tabla *directorio* de la base de datos de nombre *pruebas* utilizando cada uno de los métodos disponibles, que iremos explicando al darles uso.

Como base para la impresión, vamos utilizar un fichero cuyo contenido es el siguiente:

```
1<?php
2include ("includes/config.php");
3include ("includes/funciones.php");
4
5//nos conectamos a mysql
6$cnx = conectar ();
7//consulta.
8$sql = "SELECT nick,email,url FROM directorio ORDER BY id ASC";
9$res= mysql_query($sql) or die (mysql_error());
10if( mysql_num_rows($res) >0){
11 //impresión de los datos.
12}else{
13 echo "no se obtuvieron resultados";
14}
15mysql_close($cnx);
?>
```

A partir de la línea 11 de este fichero (del que puede encontrar una copia de nombre `baseImpresionResultados.php` en la carpeta `material/cap5/php/` del CD), es el lugar en el que insertaremos las distintas formas de impresión de datos.

Impresión de resultados usando un *array*

Mediante la función `mysql_fetch_row` se procesa el resultado como un *array*, donde cada uno de los índices del *array* es una columna del resultado. Para conseguir esto, es útil saber la cantidad de columnas de la tabla (en el caso de que en la consulta se hayan seleccionado todas ellas), siendo el orden en el que aparezcan los campos tras la consulta lo que defina el índice del *array*:

```

1<?php
2include ("includes/config.php");
3include ("includes/funciones.php");
4
5//nos conectamos a mysql
6$cnx = conectar ();
7//consulta.
8$sql = "Select nick,email,url FROM directorio ORDER BY id ASC";
9$res= mysql_query($sql) or die (mysql_error());
10if( mysql_num_rows($res) >0){
11 //impresión de los datos.
12 while ($fila = mysql_fetch_row($res)) {
13     echo "nombre: ". $fila[0]. ", email: " . $fila[1] . ", url: " .
14 $fila[2] . "<br>";
15 }
16 }else{
17 echo "no se obtuvieron resultados";
18 }
19mysql_close($cnx);
20?>

```

Observe cómo en la línea 12 se accede a las columnas en el resultado mediante la expresión `$fila[número_columna]`. Posteriormente, se concatenan las cadenas de texto obtenidas y un salto de línea mediante el *tag* `
` de HTML para imprimir un resultado por cada fila. Si escribe el código en un nuevo documento de texto o modifica el fichero base para añadir código a partir de la línea 11 y lo guarda con el nombre de

`mysql_fetch_row.php` en la carpeta `phpmysql` de su servidor (también puede utilizar el documento `material/cap5/php/mysql_fetch_row.php` del CD), al abrir el documento en su navegador tecleando `http://localhost/phpmysql/mysql_fetch_row.php`, obtenemos una impresión en pantalla como la de la figura 5.2.

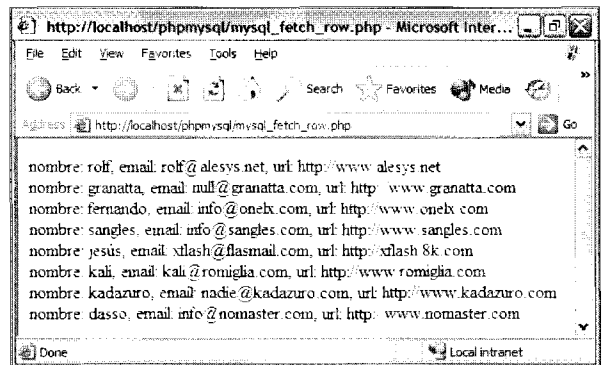


Figura 5.2.

*Impresión de resultados usando
mysql_fetch_row.*

Impresión de resultados usando un array de lista

Para imprimir los resultados utilizando un *array* de lista, puede utilizar la función `mysql_fetch_array`, donde cada columna de la tabla presente en la consulta es un índice del *array*. Esta circunstancia es de gran utilidad cuando hemos de seleccionar todas las columnas de la tabla, ya que no hemos de

recordar los índices de las columnas sino que es suficiente con recordar los nombres de las mismas. El siguiente código (que puede encontrar en el documento `material/cap5/php/mysql_fetch_array.php` del CD) selecciona las columnas `nick`, `email` y `url` en la consulta SQL, siendo esos nombres los que se utilizando como índices del `array` a la hora de extraer la información (línea 13):

```
1<?php
2include ("includes/config.php");
3include ("includes/funciones.php");
4
5//nos conectamos a mysql
6$cnx = conectar ();
7//consulta.
8$sql = "Select nick,email,url FROM directorio ORDER BY id ASC";
9$res= mysql_query($sql) or die (mysql_error());
10if( mysql_num_rows($res) >0){
11 //impresión de los datos.
12 while ($fila = mysql_fetch_row($res)) {
13     echo "nombre: " . $fila['nick'] . ", email: " . $fila['email'] . ",
url: " . $fila['url'] . "<br>";
14 }
15}else{
16 echo "no se obtuvieron resultados";
17}
18mysql_close($cnx);
?>
```

Si ahora crea un documento con este código, lo guarda en la carpeta `phpmysql` de su servidor con el nombre `mysql_fetch_array.php` y lo abre en su navegador tecleando `http://localhost/phpmysql/mysql_fetch_array.php`, obtendrá la información deseada del mismo modo en que se muestra en la figura 5.3.

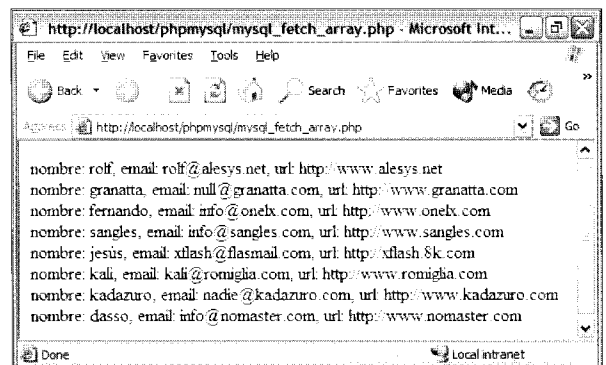


Figura 5.3.

*Impresión de resultados usando
`mysql_fetch_array`.*

Impresión de resultados usando un objeto

Utilizando la función

`mysql_fetch_object`, cada columna de la fila es una propiedad de un objeto. En los ejemplos anteriores, las variables a las que se asignaba el resultado actual en el bucle `while` eran estructuras de tipo `array`, por lo que se accedía a la información utilizando índices (numerales o nominales) por medio de los caracteres "[" y "]". En este caso, la variable que recoge la información es un objeto cuyas propiedades (que llevarán el nombre de las columnas) son accedidas por medio del operador `->`. Por ejemplo, la propiedad `nombre` del objeto `$dato` es accedida mediante la expresión:

```
$dato->nombre;
```

El siguiente código (que puede encontrar en el documento `material/cap5/php/mysql_fetch_object.php` del CD) extrae la información de la base de datos y la imprime utilizando un objeto:

```
1<?php
2include ("includes/config.php");
3include ("includes/funciones.php");
4
5//nos conectamos a mysql
6$cnx = conectar ();
7//consulta.
8$sql = "Select nick,email,url FROM directorio ORDER BY id ASC";
9$res= mysql_query($sql) or die (mysql_error());
10if( mysql_num_rows($res) >0){
11 //impresión de los datos.
12 while ($fila = mysql_fetch_object($res)){
13     echo "nombre: ". $fila->nick . ", email: " . $fila->email . ", url: "
14     . $fila->url . "<br>";
15 }
16 }else{
17     echo "no se obtuvieron resultados";
18 }
19mysql_close($cnx);
20?>
```

Observe cómo en la línea 13 se accede a las distintas propiedades del objeto `$fila`. Al guardar este fichero con el nombre de `mysql_fetch_object.php` en la carpeta `phpmysql` de su servidor y abrirlo posteriormente en su navegador tecleando `http://localhost/phpmysql/mysql_fetch_object.php`, obtendrá la información deseada del mismo modo en que se muestra en la figura 5.4.

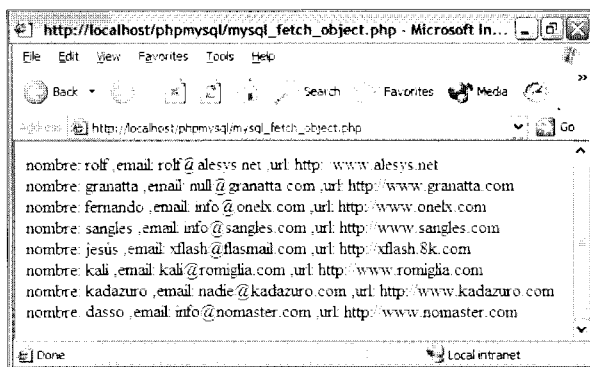


Figura 5.4.

*Impresión de resultados usando
`mysql_fetch_object`.*

La última forma que veremos de procesar los resultados es mediante el uso de la función `list` (lista) junto con las funciones `mysql_fetch_array` y `mysql_fetch_row`. De este modo, en la lista se pasan como parámetros los nombres de las variables que deseemos para cada columna, y a la hora de imprimir los resultados utilizaremos dichos nombres.



Nota: A la hora de utilizar la función `list`, recuerde siempre establecer tantos parámetros como columnas tenga la consulta SQL.

El siguiente código (que puede encontrar en el documento `material/cap5/php/mysql_fetch_list.php` del CD) extrae la información de la base de datos y la imprime utilizando conjuntamente las funciones `list` y `mysql_fetch_array`:

```
1<?php
2include ("includes/config.php");
3include ("includes/funciones.php");
4
5//nos conectamos a mysql
6$cnx = conectar ();
7//consulta.
8$sql = "Select nick,email,url FROM directorio ORDER BY id ASC";
9$res= mysql_query($sql) or die (mysql_error());
10if( mysql_num_rows($res) >0){
11 //impresión de los datos.
12 while (list($nick,$email,$url) = mysql_fetch_array($res)) {
13     echo "nombre: ". $nick . ", email: " . $email . ", url: " . $url .
"<br>";
14 }
15}else{
16 echo "no se obtuvieron resultados";
17}
18mysql_close($cnx);
?>
```

Observe en la línea 8 cómo la cantidad de variables es equivalente al número de columnas del resultado de la consulta. En la línea 12 puede observar cómo se asigna el resultado de la consulta a las variables utilizando la función `list`. Una de las ventajas de esta forma de impresión de datos es que puede mostrar los resultados en una sola línea sin necesidad de concatenar la cadena, de forma que:

```
echo "nombre: $nick, email: $email,
url: $url<br>";
```

La expresión anterior retorna el mismo resultado que la expresión que veníamos utilizando hasta ahora:

```
echo "nombre: ". $nick . ", email: "
. $email . ", url: " . $url .
"<br>";
```

Guarde este fichero con el nombre de `mysql_fetch_list.php` en la carpeta `phpmysql` de su servidor y ábralo posteriormente en su navegador tecleando `http://localhost/phpmysql/mysql_fetch_list.php` para obtener la información deseada en la forma que se muestra en la figura 5.5.

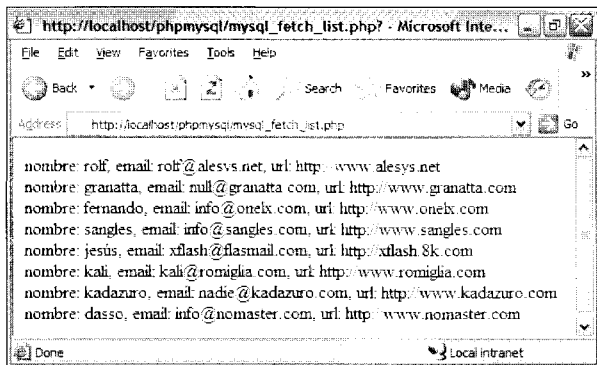


Figura 5.5.

Impresión de resultados utilizando `list` y `mysql_fetch_array`.

```
1<?php
2include ("includes/config.php");
3include ("includes/funciones.php");
4
5//nos conectamos a mysql
6$cnx = conectar ();
7//consulta.
8$sql = "Select nick,email,url FROM directorio ORDER BY id ASC";
9$res= mysql_query($sql) or die (mysql_error());
10if( mysql_num_rows($res) >0){
11 //impresión de los datos.
12 while (list($nick,$email,$url) = mysql_fetch_array($res)) {
13     echo "nombre: " . $nick . ", email: " . $email . ", url: " . $url .
14     "<br>";
15 }
16 }else{
17 echo "no se obtuvieron resultados";
18 }
19mysql_free_result($res)
20mysql_close($cnx);
21?>
```

Liberación de memoria

Si en un determinado momento se está utilizando de forma intensiva su ordenador o cree estar utilizando más de lo debido la memoria del mismo y desea "aliviarla", puede utilizar la función `mysql_free_result` para liberar memoria. La sintaxis de uso de la función es la siguiente:

```
mysql_free_result(identificador de resultado);
```

El siguiente ejemplo asigna el resultado a la variable `$res` (línea 9) y libera la memoria utilizada mediante una llamada a la función `mysql_free_result` (línea 18):

Cierre de la conexión con MySQL

PHP está configurado para cerrar las conexiones abiertas con las bases de datos y eliminar las variables utilizadas una vez que termina de ejecutar las páginas de extensión .php. Sin embargo, para asegurarnos de que realmente terminen las conexiones, puede indicarle a PHP que cierre la conexión con MySQL utilizando la función `mysql_close`, con el identificador de conexión especificado como parámetro en la llamada a la función (como se puede observar, por ejemplo, en la línea 19 del código del apartado anterior), siguiendo la sintaxis:

```
mysql_close($cnx);
```

Modificación de la información de la base de datos

En este apartado vamos a modificar la información contenida en la tabla *directorio* de la base de datos *pruebas*, lo que lograremos mediante el uso de formularios. También utilizaremos los mismos ficheros de configuración (*config.php* y *funciones.php*) que creamos anteriormente y aplicaremos todos los conceptos explicados hasta ahora de una forma más práctica, de forma que pueda comprobar cómo PHP imprime los resultados obtenidos de MySQL y cómo puede modificarlos.

Los cambios que realizaremos en la tabla son los que se podrían denominar como "genéricos" a la hora de manipular información ya almacenada:

- Listado de los registros.
- Obtención de detalles de un registro.
- Modificación de un registro.
- Inserción de un nuevo registro.
- Borrado de un registro específico.

En la mayoría de casos, la forma en que se procesan formularios consiste en utilizar tres páginas distintas:

- Página con el formulario.
- Página que procesa los datos introducidos en el formulario.
- Página de información de los cambios realizados.

Sin embargo, en este apartado aprenderemos a crear documentos PHP "inteligentes", que consisten en que una sola página .php contendrá el formulario, lo procesará y posteriormente informará del éxito o fracaso de las tareas realizadas. Asimismo, recuerde que en la tabla *directorio* utilizábamos un identificador único (la columna *id*), ya que pronto veremos la utilidad de esta columna; pero, previamente, conozcamos las formas en que las variables pueden ser enviadas cuando se utilizan formularios mediante los métodos POST y GET.

Envío de variables usando POST y GET

Cuando envíe variables desde formularios puede utilizar los métodos POST y GET, siendo la diferencia entre ambos que las variables enviadas mediante GET son adjuntadas a la dirección Web que se teclea en la barra de dirección del navegador, al contrario que las variables enviadas mediante POST.

Envío de variables mediante GET

El envío de variables mediante GET se realiza de la siguiente forma:

```
http://localhost/phpmysql/  
variablesGET.php?id=1&user=Juan
```

Observe el carácter `?`, que se utiliza para indicar el fin de la dirección Web y el comienzo de las variables enviadas mediante GET y especificadas mediante el carácter `&`. Cada nombre de variable, junto con el valor de la misma, son lo que se denominan un "par", por lo que suele decirse que las variables de los formularios se obtienen en "pares".

Una forma segura para procesar los formularios consiste en procesar las variables sólo si han sido enviadas mediante el método esperado. Las variables enviadas mediante GET pueden ser escritas directamente en la barra de dirección de su navegador, por lo que no debe utilizarlas cuando desee realizar operaciones "delicadas" (esto es, que contengan información que no desee que un usuario pueda ver, como por ejemplo, el par `nombre_de_usuario` y `contraseña` para acceder a un documento concreto de un sitio Web).

Vamos a realizar un pequeño ejemplo, enviando primero las variables mediante GET y posteriormente mediante POST. Abra un nuevo documento de texto y escriba lo siguiente:

```
<?php  
echo "variable id con valor: " . $_GET['id'];  
?>
```

Guarde el documento con el nombre `variablesGET.php` en la carpeta `phpmysql` de su servidor local en la que hemos estado trabajando y ábralo tecleando su dirección (`http://localhost/phpmysql/variablesGET.php`).

El mensaje que aparecerá en su pantalla será el siguiente:

```
variable id con valor:
```

Como puede observar en el código PHP, se está esperando una variable de nombre `id` enviada mediante GET que, cuando sea recibida, será impresa en su pantalla como parte del resultado. Puesto que no hemos especificado el valor de `id` en la llamada al documento `.php`, no se muestra ningún valor al generarse los resultados. Podemos configurar el documento PHP para que no se imprima nada hasta que la variable `$_GET['id']` exista (haciendo uso de la función `isset`), como puede ver en el código disponible en el documento `material/cap5/php/variablesGET.php` del CD:

```
<?php
if(isset($_GET
echo "variable id con valor: " . $_GET['id'];
?>
```

Modifique su documento `variablesGET.php` con el código anterior, guárdelo y abra la página nuevamente en su navegador, comprobando que PHP no imprime nada en pantalla.

Ahora abra de nuevo el documento en su navegador, pero agregando lo siguiente a la dirección de la página:

```
"?id=hola"
```

Con lo que la barra de dirección de su navegador ha de contener la cadena: `"http://localhost/phpmysql/variablesGET.php?id=hola"`

Pulse la tecla **Intro** para procesar el documento y obtendrá en su pantalla el siguiente texto, tal y como se muestra en la figura 5.6:

```
variable id con valor: hola
```

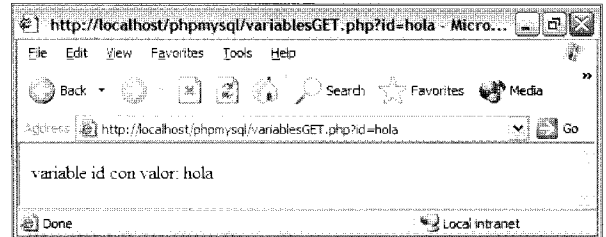


Figura 5.6.

Impresión de variables enviadas mediante GET.

En este caso, el documento PHP recibió un valor para la variable `id` mediante el método GET, puesto que lo escribimos en la barra de dirección del navegador.

A continuación realizaremos la misma operación pero utilizando un formulario para especificar el valor de la variable `id`. Cree un nuevo documento de texto de nombre `variablesGET_b.php` (puede encontrar una copia del mismo en el directorio `material/capitulo5/php/` del CD) y añada este código:

```
<?php
if(isset($_GET['id'])) {
    echo "variable id con valor: " . $_GET['id'];
}
?>

<form method="GET" action="<? echo $_SERVER['PHP_SELF'];?>">
<input type="text" name="id"><br>
<input type="submit" name="submit" value="enviar">
</form>
```

Hemos añadido al documento un campo de texto de nombre `id` y un botón de tipo `SUBMIT` para enviar datos. La forma en que se envían las variables del formulario es `GET`, y como acción especificamos lo siguiente:

```
$_SERVER['PHP_SELF']
```

`PHP_SELF` es una variable de servidor que retorna la ruta hasta el fichero, lo que nos indica que este documento, que es el que contiene el formulario, también será el que procese la información. Guarde el fichero y ábralo en su navegador tecleando `http://localhost/phpmysql/variablesGET_b.php`.

En este momento podrá ver el campo de texto y el botón, introduzca el texto que desee y presione `SUBMIT`, obteniendo un resultado que ha de ser similar al de la figura 5.7.

Observe cómo las variables son visibles (ya que aparecen en la barra de dirección del navegador) y cómo se ha añadido una nueva variable de nombre `submit`, que se corresponde con el nombre del botón, y cuyo valor es en este caso "enviar". Posteriormente veremos la utilidad que podemos aplicar a esta variable cuando procesemos un formulario.

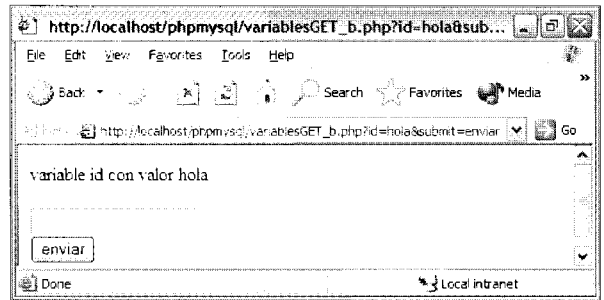


Figura 5.7.

Impresión de variables enviadas mediante GET vía formulario.

Podemos concluir entonces que el envío de variables mediante `GET` puede obtenerse escribiendo el nombre de las variables en la barra de dirección o bien enviándolas desde un formulario. Más tarde las usaremos en forma de hipervínculo.

Envío de variables mediante POST

Para probar el formulario usando `POST` como método de envío, modifique el código del fichero anterior para que contenga lo siguiente:

```
<?php
if(isset($_POST['id'])) {
    echo "variable id con valor: " . $_POST['id'];
}
?>

<form method="POST" action="<? echo $_SERVER['PHP_SELF'];?>">
<input type="text" name="id"><br>
<input type="submit" name="submit" value="enviar">
</form>
```

Guarde el archivo con un nuevo nombre (`variablesPOST.php`) en la carpeta `phpmysql` de su servidor local y ábralo en su navegador tecleando `http://localhost/phpmysql/variablesPOST.php`.

En esta ocasión, PHP está esperando a que la variable `id` sea enviada mediante `POST`, por lo que para comprobar que PHP reconoce el método con el que se envían las variables, teclee el valor para la variable `id` en la barra de dirección del navegador de la misma forma en que lo hicimos anteriormente con `GET`:

```
http://localhost/phpmysql/  
variablesPOST.php?id=hola
```

Al pulsar la tecla **Intro**, puede observar que PHP no imprime resultados debido a que la variable no es pasada mediante el método que se espera (figura 5.8)

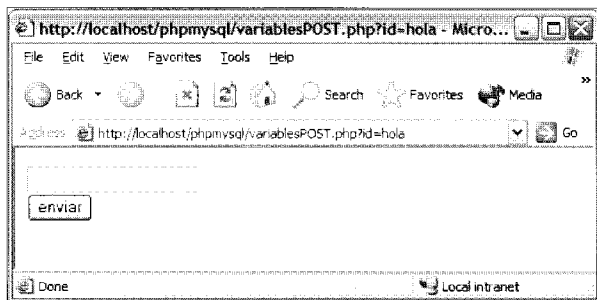


Figura 5.8.

Variables enviadas mediante `GET` ignoradas cuando se esperan vía `POST`.

Si en cambio introduce un texto en el campo de texto del formulario y pulsa el botón `SUBMIT` para enviar los datos, podrá ver cómo el resultado sí es impreso al procesar el formulario.

El uso de variables mediante `POST` es más seguro, puesto que ni las variables ni los valores que almacenan son visibles en la barra de dirección del navegador, por lo que una vez enviado el formulario no pueden ser modificadas.

Listado de registros

La forma más fácil de conocer los registros de que disponemos en una tabla de la base de datos es listarlos para, a partir de entonces, saber qué registro queremos conocer en detalle, modificar o borrar. Este tipo de tarea suelen llevarla a cabo páginas PHP que disponemos como índice del proyecto: en este caso, como en los venideros, cada fichero dispondrá de un nombre acorde con la actividad que desarrolla.

Para crear la página que realizará los listados necesitaremos realizar una selección de todos los registros que contengan las columnas `nick` e `id` de la tabla *directorio*. Estos datos los mostraremos dentro de una tabla generada con *tags* HTML y, además, estableceremos hipervínculos a las páginas que nos servirán para ver en detalle, editar y/o eliminar la información de cada una de las filas de la tabla, utilizando como parámetro el valor almacenado en el campo `id` del registro.

Tabla 5.1.
Estructura de la página de listado.

<i>ID de registro</i>	<i>Nombre</i>	<i>Acción</i>
Id de registro1	Nick de persona1	ver editar eliminar
...
Id de registroN	Nick de personaN	ver editar eliminar

Para generar tanto las páginas que realicen el listado como las siguientes combinaremos texto HTML con PHP (incluyendo las consultas SQL de éste a la base de datos y la impresión de resultados); especificaremos "dentro de PHP" cuando el código que tecleemos se encuentre entre las etiquetas PHP (<? y ?>) y "fuera de PHP" cuando hagamos referencia a texto normal, por lo general, escrito en lenguaje HTML.

Crearemos un nuevo documento de texto e incluiremos los archivos de configuración para comenzar:

```
<?
include ("includes/config.php");
include ("includes/funciones.php");
?>
```

A continuación, fuera de PHP, comenzaremos a dibujar la tabla que almacenará la información, junto con lo que serán las cabeceras de la misma. Como puede notar observando la tabla 5.1, se requieren cinco columnas: id, nombre, ver, editar y eliminar, estando las tres últimas agrupadas bajo la misma cabecera de la tabla (la que lleva por título *acción*). El código HTML que genera esta estructura sería el siguiente:

```
<table width="500" border="1" cellpadding="0" cellspacing="0">
<tr>
  <td>id</td>
  <td>Nombre</td>
  <td colspan="3" align="center" >acci&oacute;n</td>
</tr>
```

Las líneas anteriores generan la declaración de la tabla (etiqueta <table . . .>) junto con su primera fila (<tr> . . . </tr>), en la que establecemos los textos de la cabecera. Cuando hayamos de imprimir los distintos regis-

tros tras la consulta mostraremos uno por fila, pero antes necesitaremos conectarnos a la base de datos mediante PHP utilizando la función que hemos declarado en el fichero `funciones.php` (*conectar*), guardando el identificador de conexión en una variable de nombre `$cnx`, para posteriormente realizar la consulta SQL a MySQL:

```
<?
//nos conectamos a mysql
$cnx = conectar ();
```

Como comentamos anteriormente, sólo necesitaremos las columnas `nick` e `id` a la hora de realizar la consulta, obteniendo de ambas todos los registros en la tabla `directorio`. La consulta SQL se podría formular de la siguiente forma:

```
$sql = " SELECT id,nick FROM
directorio ORDER BY id ASC";
```



Nota: Recuerde que no es lo mismo seleccionar todas las columnas de una tabla ("SELECT * FROM ...") que seleccionar todos los registros, ya que cuando sólo necesitamos algunos se utilizan los denominados límites ("...LIMIT 0, 10"), que ya vimos en el capítulo anterior.

Observe cómo en la consulta SQL se ordenan de forma ascendente los resultados obtenidos de seleccionar las columnas `id` y `nick`. A continuación realizamos la consulta a MySQL almacenando el indicador de resultado en la variable `$res`, especificando que se impriman los errores en caso de que se produzca alguno:

```
$res= mysql_query($sql) or die
(mysql_error());
```

Realizada la consulta, revisamos si existen resultados de la misma, para lo que comprobamos si el número de filas almacenado en la variable `$res` es mayor que cero:

```
if( mysql_num_rows($res) >0){
```

Y si efectivamente comprobamos que existe más de una fila (lo que quiere decir que hay resultados devueltos por la consulta) procedemos a la impresión de los datos en la pantalla, utilizando para ello un bucle de tipo `while`; puesto que sólo tenemos dos columnas seleccionadas, la forma más sencilla de procesar los datos es asignar cada columna a una variable por medio de la función `list`:

```
while (list($id,$nick) =
mysql_fetch_array($res)) {
```

A partir de ahora, el resultado de la fila en que se encuentre el bucle `while` que usamos para recorrer las columnas `id` y `nick` se encontrará en las variables `$id` y `$nick` que hemos asignado. Las dos primeras celdas de la tabla HTML son las que contienen los valores de las variables, así que las imprimimos dándole el formato de celda de tabla HTML (`<tr>`
`<td>...</td> ...<td> ...</td></`

`tr`), donde el *tag* `<tr>` sirve para especificar el comienzo de una fila y los *tags* `<td>` sirven para especificar el comienzo de una de las celdas de la fila):

```
echo "<tr><td>$id</td>\n";
echo "<td>$nick</td>\n";
```



Nota: El carácter de salto de línea (`\n`) que se agrega al final de cada llamada a la función `echo` se utiliza para que el código HTML resulte legible una vez impresa la página.

Las tres siguientes celdas que se añaden a la fila han de contener hipervínculos a los documentos `ver.php`, `editar.php` y `eliminar.php`, por lo que para indicarles a éstas el registro que se quiere ver o editar utilizaremos el método `GET`, para pasar el valor de la columna `id` en la llamada al documento PHP. Por ejemplo, si la variable `id` contiene el valor 5, en el hipervínculo que apunta hacia el documento `ver.php` haremos referencia al valor de `id` de la siguiente forma:

```
ver.php?id=5
```

lo que en HTML se genera así:

```
<a href="ver.php?id=5">ver</a>
```

y si utilizamos PHP, como es nuestro caso, habremos de generarlo de este modo:

```
echo "<a href='ver.php?id=$id'>ver</a>";
```

Si deseamos imprimir comillas dobles (" y ") dentro del código HTML, podemos conseguirlo anteponiendo el carácter `"\"` cada vez que aparecen las comillas en el texto:

```
echo "<a href=\"ver.php?id=$id\">ver</a>";
```

O bien de esta otra forma:

```
echo "<a href='ver.php?id=$id'>ver</a>";
```

Sea cual sea la opción que elija, se imprimirá el mismo resultado. Recordemos también que hemos de imprimir los *tags* (`<td> ... </td>`) que especifiquen que el texto con el vínculo se encuentran en una celda de la fila. Al final, la impresión de las tres celdas conteniendo los textos con hipervínculo se realizará así:

```
echo "<td><a
href='ver.php?id=$id'>ver</a></
td>\n";
echo "<td><a
href='editar.php?id=$id'>editar</
a></td>\n";
echo "<td><a
href='eliminar.php?id=$id'>eliminar</
a></td></tr>\n";
```

Observe que en la última línea se agrega la etiqueta que cierra la fila HTML (`</tr>`). En caso de que no se hayan obtenido resultados, es conveniente imprimir un mensaje que informe de dicha incidencia:

```
}else{
    echo "<td colspan='5'
align='center'>no se obtuvieron
resultados</td>";
}
```



Nota: La razón por la que se añade `colspan='5'` en la declaración de la celda es la de que estamos imprimiendo texto dentro de una tabla HTML que dispone de cinco celdas por fila.

Cerramos la conexión a MySQL y salimos de PHP:

```
mysql_close($cnx);  
?>
```

Y por último, cerramos la tabla HTML:

```
</table>
```

El código completo del documento (que puede encontrar en `el material/cap5/php/listado.php` del CD) es el siguiente:

```
1<?php  
2include ("includes/config.php");  
3include ("includes/funciones.php");  
4?>  
5<table width="500" border="1" cellpadding="0" cellspacing="0">  
6<tr>  
7 <td>id</td>  
8 <td>Nombre</td>  
9 <td colspan="3" align="center" >acci&oacute;n</td>  
10 </tr>  
11<?  
12//nos conectamos a mysql  
13$cnx = conectar ();  
14//consulta.  
15$sql = "Select id,nick FROM directorio ORDER BY id ASC";  
16$res= mysql_query($sql) or die (mysql_error());  
17  
18if( mysql_num_rows($res) >0){  
19 //impresión de los datos.  
20 while (list($id,$nick) = mysql_fetch_array($res)) {  
21 echo "<tr><td>$id</td>\n";  
22 echo "<td>$nick</td>\n";  
23 echo "<td><a href='ver.php?id=$id'>ver</a></td>\n";  
24 echo "<td><a href='editar.php?id=$id'>editar</a></td>\n";  
25 echo "<td><a href='eliminar.php?id=$id'>eliminar</a></td></tr>\n";  
  
26 }  
27}else{  
28 echo "<td colspan='5' align='center' >no se obtuvieron resultados</td>";  
29}  
30mysql_close($cnx);  
31?>  
32</table>
```

Guarde este documento con el nombre `listado.php` en la carpeta `phpmysql` de su servidor local y ábralo en su navegador tecleando `http://localhost/phpmysql/listado.php`. Los resultados que debe ver en su pantalla han de ser como los de la figura 5.9.

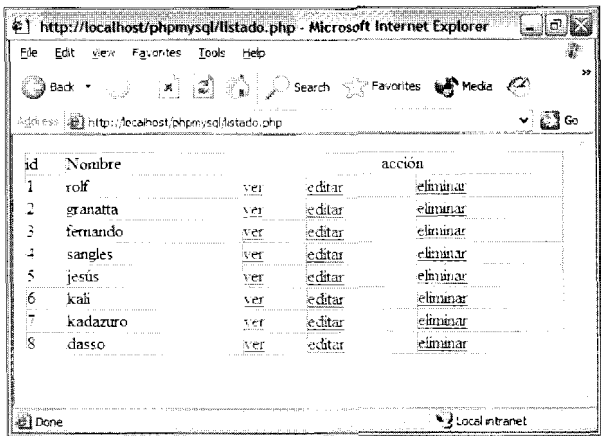


Figura 5.9.
Fichero `listado.php` mostrando resultados en pantalla.

Obtención de detalles de un registro

El documento que nos mostrará los detalles de un registro es una selección de todas las columnas de la tabla `directorio`. La variable `id`, enviada mediante GET, nos ayudará a saber cuál es el registro que hay que mostrar y, una vez realizada la consulta a la base de datos, mostraremos la información en una tabla HTML con una estructura como la mostrada en la tabla 5.2.

Tabla 5.2.
Estructura de la página de detalle.

ID de registro	id
Nombre	nombre apellido
nick	nick
email	email
Web site	url

Este tipo de documentos podemos dividirlos en dos secciones:

- Si la variable `id` enviada mediante GET no existe, no se puede continuar.
- Si existen resultados, los muestra. En caso contrario, se muestra un mensaje de ello.

Esta última sección ya sabemos cómo implementarla; la primera la desarrollaremos a continuación. Como vimos en uno de los apartados anteriores, la función `isset` se utiliza para saber si una variable existe (está definida) o no, y su sintaxis es:

```
isset($miVariable)
```

Si `$miVariable` está definida, la función `isset` retorna el valor `true` (verdadero), pero lo que deseamos consultar es si no está definida, para lo que utilizamos el operador `!` (NOT):

```
!isset($miVariable)
```

En caso de que la variable no exista, lo que haremos será retornar a la página de listado:

```
if(!isset($mivariable)){
    //redireccionar
}
```

Pero, ¿qué ocurriría si la variable está definida pero no tiene un valor asignado? La página no accederá al redireccionamiento y seguirá ejecutándose. Para evitar este hecho utilizaremos la función `empty`, que sirve para comprobar si la variable está definida y además contiene algún valor (no está vacía). Su sintaxis es la siguiente:

```
empty($miVariable)
```

Por tanto, el uso que daremos a la función será:

```
if(empty($mivariable)){
    //redireccionar
}
```

Para redireccionar y retornar a la página de listado, usaremos las siguientes dos líneas de código:

```
header("Location: listado.php");
exit;
```

El documento `ver.php` comenzará verificando la existencia y valor de la variable `id` y comprobará que ésta haya sido enviada mediante el método GET. Si todo es correcto, se incluyen nuestros archivos de configuración y funciones, y en caso contrario se redireccionará al usuario a la página de listado (todas estas acciones se declaran dentro de PHP).

```
<?php
//si no hay id, no puede seguir.
if(empty($_GET['id'])){
    header("Location: listado.php");
    exit;
}
include ("includes/config.php");
include ("includes/funciones.php");
?>
```

A continuación, construimos en HTML la tabla que contendrá la información:

```
<table width="400" border="1"
cellpadding="0" cellspacing="0">
```

Establecemos una conexión con la base de datos, entrando de nuevo en PHP:

```
<?
//nos conectamos a mysql
$cnx = conectar ();
```

Recuerde, a la hora de realizar la consulta SQL, que deseamos seleccionar todas las columnas donde el identificador único tenga el mismo valor que el de la variable `id` enviada mediante GET:

```
$sql = "SELECT * FROM directorio
WHERE id=".$_GET['id'];
```

Posteriormente, efectuamos la consulta SQL almacenando el identificador de resultado en una variable de nombre `$res` y especificando que imprima los errores si se producen:

```
$res= mysql_query($sql) or die
(mysql_error());
```

Comprobamos el número de filas de la variable `$res` para saber si hay resultados o no:

```
if( mysql_num_rows($res) >0){
```

Si encontramos resultados, comenzamos a procesarlos asignando el resultado que retorna la función `mysql_fetch_array` (un *array* de lista con los nombres de las columnas) a una variable de nombre `$fila`:

```
while ($fila =
mysql_fetch_array($res)) {
```

En esta ocasión no saldremos de PHP, y la siguiente sección la generaremos mediante HTML, insertando pequeños fragmentos de código PHP cuando deseemos imprimir los datos de la columna. Por ejemplo, si vamos a imprimir la columna `id`, utilizaríamos:

```
<tr>
  <td>Id</td>
  <td><>
  <td><? echo $fila['id'];?></td>
</tr>
```

En PHP se realizaría de la misma forma que vimos con anterioridad:

```
echo "<tr><td>Id</td>
  <td><td>". $fila['id'] . "</td></tr>";
```

Insertar pequeños fragmentos de código PHP dentro de un documento escrito en HTML nos ayuda visualmente, ya estemos utilizando un editor de texto o un editor de tipo WYSIWYG para generar los documentos .html. Puede observar una comparación "visual" entre ambos métodos (`ver.php`, con fragmentos de PHP dentro de HTML, y `verb.php`, completamente escrito en PHP) en la figura 5.10.

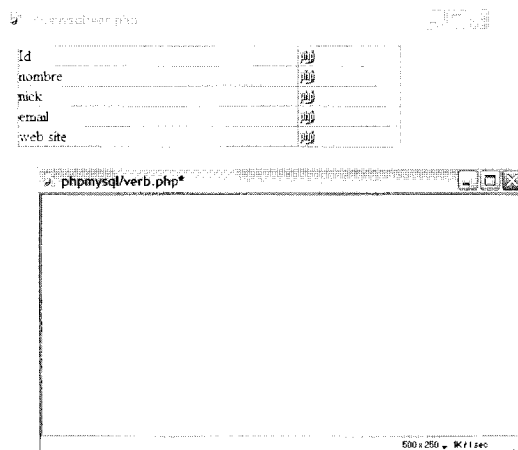


Figura 5.10.

Código PHP embebido en HTML y código PHP.

Cuando anteriormente creamos el documento `listado.php`, el código HTML fue impreso por medio de PHP. En esta ocasión el código PHP será embebido o incrustado en el código HTML, de modo que dispone de varias opciones para trabajar, todas ellas perfectamente válidas. Elija aquella que más cómoda le resulte.

La impresión de los datos consiste en una columna de la consulta por cada fila de la tabla HTML, donde una de las celdas se utiliza para imprimir el nombre y la otra para imprimir el valor de la columna (en el resultado de la consulta), teniendo en cuenta que tanto el nombre como el apellido se mostrarán en una sola celda:

```
<td><? echo $fila['nombre'] . " ".
  $fila['apellido'];?></td>
```

Ya que estamos siguiendo la estructura de tabla que especificamos en la tabla 5.2. El código para imprimir los datos será el siguiente:

```
<tr>
  <td>Id</td>
  <td><? echo $fila['id'];?></td>
</tr>
<tr>
  <td>nombre</td>
  <td><? echo $fila['nombre']. " ". $fila['apellido'];?></td>
</tr>
<tr>
  <td>nick</td>
  <td><? echo $fila['nick'];?></td>
</tr>
<tr>
  <td>email</td>
  <td><? echo $fila['email'];?></td>
</tr>
<tr>
  <td>web site</td>
  <td><? echo $fila['url'];?></td>
</tr>
```

Cuando hayamos terminado de imprimir los datos, cerraremos el bucle `while` y especificaremos el mensaje alternativo que ha de mostrarse en caso de que no se hayan encontrado resultados, utilizando para ello código PHP:

```
<?
}
}else{
  echo "<tr><td colspan='2' align='center'>no se obtuvieron resultados</td></tr>";
}
```

Para finalizar la construcción de la página, cerramos tanto la conexión con MySQL como la tabla HTML:

```
mysql_close($cnx);
?>
</table>
```

El código completo de la página `ver.php` (que puede encontrar en el documento `material/cap5/php/ver.php` del CD) es el siguiente:

```

1<?php
2//si no hay id, no puede seguir.
3if(empty($_GET['id'])){
4    header("Location: listado.php");
5    exit;
6}
7include ("includes/config.php");
8include ("includes/funciones.php");
9?>
10<table width="400" border="1" cellpadding="0" cellspacing="0">
11<?
12//nos conectamos a mysql
13$cnx = conectar ();
14//consulta.
15$sql = "SELECT * FROM directorio WHERE id=".$_GET['id'];
16$res= mysql_query($sql) or die (mysql_error());
17
18if( mysql_num_rows($res) >0){
19    //impresión de los datos.
20    while ($fila = mysql_fetch_array($res)) {
21        ?>
22<tr>
23    <td>Id</td>
24    <td><? echo $fila['id'];?></td>
25</tr>
26<tr>
27    <td>nombre</td>
28    <td><? echo $fila['nombre']. " ". $fila['apellido'];?></td>
29</tr>
30<tr>
31    <td>nick</td>
32    <td><? echo $fila['nick'];?></td>
33</tr>
34<tr>
35    <td>email</td>
36    <td><? echo $fila['email'];?></td>
37</tr>
38<tr>
39    <td>web site</td>
40    <td><? echo $fila['url'];?></td>
41</tr>
42    <?
43    }
44}else{
45    echo "<tr><td colspan='2' align='center'>no se obtuvieron resultados</td></tr>";
46}
47mysql_close($cnx);
48?>
49</table>

```

Guarde este documento con el nombre
 ver.php en la carpeta phpmysql de su
 servidor local y abra de nuevo en su navega-
 dor el documento listado.php tecleando

<http://localhost/phpmysql/listado.php>.
 Cuando presione cada uno de los vínculos de
 nombre ver (junto a cada uno de los nombres
 listados) en la tabla HTML, será enviado al

documento `ver.php` que acabamos de realizar, mostrando los datos asociados al registro elegido, como puede ver en la figura 5.11 datos obtenidos tras pulsar el enlace `ver` para obtener más información sobre Rolf Ruiz.

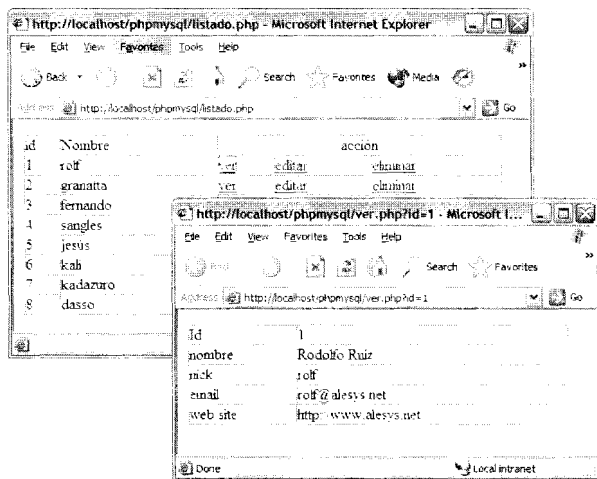


Figura 5.11.

Visualización de los detalles del registro cuyo id es 1.

Modificación de un registro

Los pasos que realizaremos para modificar los contenidos de un registro de la tabla *directorio* son los siguientes:

- Enviaremos mediante GET el identificador del registro.
- Mostraremos la información actual que contiene el registro.
- Construiremos un formulario para actualizar la información.
- Una vez enviado el formulario, actualizaremos la información.

El identificador de registro que pasamos mediante GET a las páginas `ver.php`, `editar.php` y `eliminar.php` es el identificador que utilizamos para referenciar el registro del que deseamos obtener información tras realizar la consulta SQL.

Para poder editar la información crearemos un formulario cuyos campos contengan los datos del registro. Estos campos serán editables a excepción del campo único, cuyo valor no ha de ser modificado pero sí mostrado para que sepamos sobre qué registro estamos aplicando los cambios, por lo que utilizaremos un campo oculto en lugar de un campo de texto.

El formulario, al ser enviado, actualizará los datos. Puesto que vamos a realizar todas las acciones en un solo documento, hemos de determinar cuándo se efectuarán cada una de ellas:

- Si no existe la variable enviada mediante GET, redireccionamos al usuario al documento `listado.php`.
- Si existe dicha variable, realizamos la consulta.
- Si el formulario es enviado, dispondremos de una variable de nombre `submit` que nos servirá para saber que podemos actualizar la información.

El orden en que el código estará dispuesto en el documento PHP será el siguiente:

- Actualización de los datos.
- Verificación del valor de `id`.
- Despliegue de información en el formulario.



Nota: El orden en que el código está dispuesto en el documento PHP no es el mismo orden en el que el código se ejecutará, ya que la actualización de datos no se llevará a cabo hasta que el formulario haya sido enviado, y éste no puede contener datos hasta que se haya verificado la existencia y valor de `id`.

Abra un nuevo documento de texto e incluya primeramente los archivos de configuración que hemos venido utilizando en los distintos ejemplos:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");
```

Ahora crearemos la sección de actualización de datos; nos centraremos por el momento en la consulta de los datos y en mostrar estos últimos en el formulario.



Nota: Recordemos que falta por realizar la sección en la que se van a actualizar los datos y que debe colocarse justamente después de la inclusión de los archivos de configuración.

Comprobamos si existe la variable `id` enviada mediante `GET`. De no ser así, redireccionamos al usuario al documento `listado.php`:

```
//si no hay id, no puede seguir.
if(empty($_GET['id'])) {
    header("Location: listado.php");
    exit;
}
```

Posteriormente, establecemos la conexión con MySQL:

```
$cnx = conectar ();
```

Especificamos la consulta SQL teniendo en cuenta que para editar cualquier campo del registro hemos de seleccionar todas las columnas de la tabla donde el valor del identificador único coincida con el de la variable `id` enviada mediante `GET`:

```
$sql = "SELECT * FROM directorio
WHERE id=".$_GET['id'];
```

Y realizamos la consulta configurando que se muestren los errores en caso de que se produzca alguno:

```
$res= mysql_query($sql) or die
(mysql_error());
```

Si existen resultados (el número de filas almacenado en la variable `$res` es mayor que cero), extraemos los datos para procesarlos:

```
if( mysql_num_rows($res) >0){
//si hay resultados hacemos la forma.
?>
```

En este momento hemos cerrado la etiqueta de código PHP, con lo que el código que dispondremos a continuación es HTML, mediante el que imprimiremos la etiqueta del formulario y la cabecera de la tabla que contendrá los campos de texto con la información de la base de datos:

```
<form name="form1" method="post" action="<?echo $_SERVER['PHP_SELF'];?>">
<table width="400" border="1" cellpadding="0" cellspacing="0">
```

Como dijimos anteriormente, esta página se encargará de procesar la información enviada por el formulario, por lo que establecemos como acción para el mismo la variable `$_SERVER['PHP_SELF']`, que se encargará de referenciar la ruta de este archivo para que el formulario sea enviado a la misma página en que se declara.

Abrimos de nuevo las etiquetas PHP tan sólo para crear un bucle de tipo `while`, puesto que las celdas de la tabla y los campos de texto los realizaremos mediante HTML, embebiendo el código PHP con los datos de las columnas cuando sea necesario:

```
<?
//impresión de los datos.
while ($fila = mysql_fetch_array($res)) {
?>
```

La tabla en la que mostraremos los campos de texto del formulario es similar a la que utilizamos cuando construimos el documento `ver.php`, con la salvedad de que en esta ocasión la información se mostrará en campos de texto editables (excepto el valor de la columna `id`, que se imprimirá en texto plano).

Tabla 5.3.
Estructura página editar.

ID	id
Nombre	[nombre]
Apellido	[apellido]
nick	[nick]
email	[email]
Web site	[url]

Para generar la tabla y el formulario en la forma indicada, debemos incluir la etiqueta del campo de texto en cada celda de la tabla que contenga datos:

```
<input name="{nombre}" type="text" id="{nombre}">
```

Y, posteriormente, embeber el código PHP cuando queramos imprimir el valor de la columna:

```
<input name="{nombre}" type="text" id="{nombre}" value="<? echo $fila['{nombre}'];?>">
```

Finalmente, el formato de cada una de las columnas se mostrará como sigue:

```
<tr>
  <td>{nombre}</td>
  <td><input name="{nombre}" type="text" id="{nombre}" value="<? echo $fila['{nombre}'];?>"></td>
</tr>
```

Además necesitaremos un campo de texto oculto con el valor del campo id, pudiendo añadir una impresión en texto del valor si se desea:

```
<tr>
  <td>Id</td>
  <td><input name="id" type="hidden" id="id" value="<?echo $fila['id'];?>"><? echo $fila['id'];?></td>
</tr>
```

El resto de las columnas se muestran por pantalla de forma uniforme:

```
<tr>
  <td>nombre</td>
  <td><input name="nombre" type="text" id="nombre" value="<? echo $fila['nombre'];?>"></td>
</tr>
<tr>
  <td>apellido</td>
  <td><input name="apellido" type="text" id="apellido" value="<? echo $fila['apellido'];?>"></td>
</tr>
<tr>
  <td>nick</td>
  <td><input name="nick" type="text" id="nick" value="<? echo
```

```

$fila['nick'];?>"></td>
</tr>
<tr>
    <td>email</td>
    <td><input name="email" type="text" id="email" value="<? echo
$fila['email'];?>"></td>
</tr>
<tr>
    <td>web site</td>
    <td><input name="url" type="text" id="url" value="<? echo $fila['url'];?>"></td>
</tr>

```

Para añadir el botón SUBMIT al formulario declararíamos la celda de la siguiente forma:

```

<tr>
    <td>&nbsp;</td>
    <td align="right"><input type="submit" name="submit" value="enviar"></td>
</tr>

```

Es importante que especifique submit como nombre para el botón, ya que es el dato que vamos a verificar cuando estemos realizando la actualización de los datos. Posteriormente, cerramos la tabla y el formulario HTML:

```

</table>
</form>

```

De nuevo dentro de PHP, cerramos el bucle de tipo while y especificamos un mensaje alternativo en caso de que no haya resultados tras la consulta. En esta ocasión, el mensaje no necesita mostrarse conjuntamente con código HTML, ya que la tabla sólo se imprime a la hora de construirse el formulario:

```

    <?
    }
} else {
    echo "no se obtuvieron resultados";
}

```

Cerramos la conexión a MySQL y la etiqueta PHP:

```

mysql_close($cnx);
?>

```

Ahora vamos a construir la parte que tenemos pendiente, la actualización de los datos enviados desde el formulario. Cuando éste es enviado sabemos que entre las variables que contiene existe una de nombre submit (la que envía el botón SUBMIT) y otra de nombre id (el campo oculto). Para comenzar, comprobamos la existencia de submit:

```

if(isset($_POST['submit'])) {

```

Si es así, establecemos una conexión con MySQL:

```

    $cnx = conectar ();

```

El formato de actualización de registros en la base de datos es, como vimos en el capítulo anterior:

```
UPDATE {tabla} SET campo1='valor1',...campoN='valorN' WHERE campo = valor;
```

En nuestro caso, agregaremos los campos de uno en uno y en líneas separadas para que pueda "visualizar" el proceso de recolección de datos. Declaramos una variable de nombre `$sql` para realizar la consulta:

```
$sql = "UPDATE directorio SET ";
```

Y concatenamos el contenido de la variable con el nombre de cada columna junto con la variable asociada a ésta:

```
$sql .= "nombre = '". $_POST['nombre']. "', ";  
$sql .= "apellido = '". $_POST['apellido']. "', ";  
$sql .= "nick = '". $_POST['nick']. "', ";  
$sql .= "email = '". $_POST['email']. "', ";  
$sql .= "url = '". $_POST['url']. "'";
```

Posteriormente, añadimos una cláusula `where` para indicar qué registro ha de ser modificado (aquel en el que el valor del identificador único coincida con el de la variable `id`):

```
$sql .= " WHERE id = '". $_POST['id']. '";
```

Para finalizar, mostramos un mensaje confirmando que la actualización de datos se ha efectuado con éxito y un vínculo al documento `listado.php`.

```
echo "Registro actualizado.<br><a href='listado.php'>regresar</a>"
```

Cerramos la conexión establecida con MySQL:

```
mysql_close($cnx);
```

Y hacemos uso de la función `exit` para especificarle a PHP que deje de interpretar el documento y así no aparezca de nuevo el formulario:

```
exit;  
}
```

El código completo de la página (que puede encontrar en el documento `material/cap5/php/editar.php` del CD) es el siguiente:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");

//si la forma ha sido enviada editamos el registro.
if(isset($_POST['submit'])) {
    //nos conectamos a mysql
    $cnx = conectar ();

    $sql = "UPDATE directorio SET ";
    $sql .= "nombre = '".$_POST['nombre']."'";
    $sql .= "apellido = '".$_POST['apellido']."'";
    $sql .= "nick = '".$_POST['nick']."'";
    $sql .= "email = '".$_POST['email']."'";
    $sql .= "url = '".$_POST['url']."'";
    $sql .= " WHERE id = '".$_POST['id']'";
    $res = mysql_query($sql) or die(mysql_error());

    echo "Registro actualizado.<br><a href='listado.php'>regresar</a>"
    mysql_close($cnx);
    exit;
}

//si no hay id, no puede seguir.
if(empty($_GET['id'])) {
    header("Location: listado.php");
    exit;
}

//nos conectamos a mysql
$cnx = conectar ();

//consulta para mostrar los datos.
$sql = "SELECT * FROM directorio WHERE id='".$_GET['id']'";
$res= mysql_query($sql) or die (mysql_error());

if( mysql_num_rows($res) >0){
    //si hay resultados hacemos la forma.
    ?>
    <form name="form1" method="post" action="<?echo $_SERVER['PHP_SELF'];?>">
    <table width="400" border="1" cellpadding="0" cellspacing="0">
    <?
        //impresión de los datos.
        while ($fila = mysql_fetch_array($res)) {
            ?>
            <tr>
                <td width="150">Id</td>
                <td><input name="id" type="hidden" id="id" value="<?echo $fila['id'];?>"><?
                echo $fila['id'];?></td>
            </tr>
            <tr>
```

```

<td>nombre</td>
  <td><input name="nombre" type="text" id="nombre" value="<? echo
$fila['nombre'];?>"></td>
</tr>
<tr>
  <td>apellido</td>
  <td><input name="apellido" type="text" id="apellido" value="<? echo
$fila['apellido'];?>"></td>
</tr>
<tr>
  <td>nick</td>
  <td><input name="nick" type="text" id="nick" value="<? echo
$fila['nick'];?>"></td>
</tr>
<tr>
  <td>email</td>
  <td><input name="email" type="text" id="email" value="<? echo
$fila['email'];?>"></td>
</tr>
<tr>
  <td>web site</td>
  <td><input name="url" type="text" id="url" value="<? echo $fila['url'];?>"></
td>
</tr>
<tr>
  <td>&nbsp;</td>
  <td align="right"><input type="submit" name="submit" value="enviar"></td>
</tr>
</table>
</form>
  <?
  }
}else{
  //no hay resultados, id malo o no existe.
  echo "no se obtuvieron resultados";
}
mysql_close($cnx);
?>

```

Guarde el documento con el nombre
 editar.php en la carpeta phpmysql de su
 servidor local y abra de nuevo en su navega-

dor el documento listado.php tecleando
<http://localhost/phpmysql/listado.php>.

Seleccione el registro que desee editar y realice los cambios que desee, en un proceso que debe ser como el que se muestra en la figura 5.12.

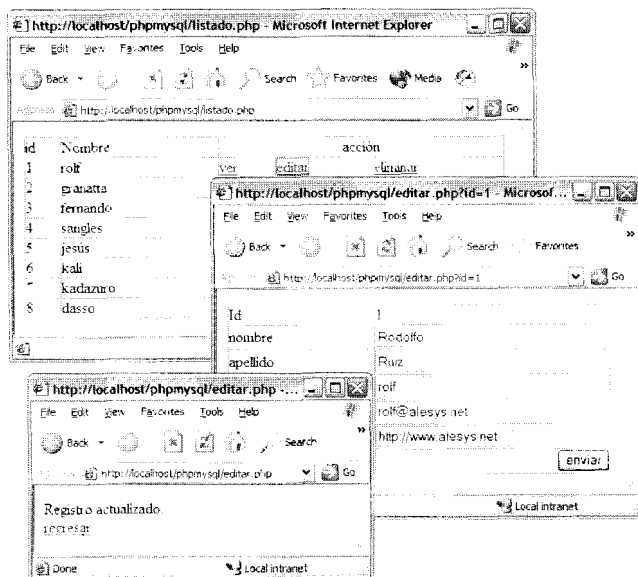


Figura 5.12.

Documento `editar.php` como formulario y resultado de edición.

Inserción de un nuevo registro

El proceso de insertar un nuevo registro es más simple que el de editarlo, ya que, aunque también necesitamos un formulario para introducir los datos, no hemos de realizar una consulta previa para llenar sus campos con los valores obtenidos de la misma. Por otra parte, tampoco necesitaremos utilizar la variable `id`, ya que al ser un registro nuevo la base de datos se encargará de asignarle el número siguiente a los ya creados (al ser ese campo de la tabla de tipo autoincremental), siendo el número asignado el que lo identificará como único en la tabla. La página PHP para realizar las inserciones sólo consta de dos partes:

- Formulario.
- Ingreso de información.

Tal y como hicimos al crear la página para editar registros, nos basaremos en la existencia de una variable llamada `submit` para saber que estamos realizando un envío de nuevos datos. Primero realizaremos el código HTML para el formulario, y posteriormente el código PHP para realizar la inserción de información. Además, puesto que no necesitamos hacer ninguna consulta para mostrar datos de la tabla *directorio*, no será preciso incluir los archivos de configuración de que disponemos en nuestro directorio `phpmysql/includes` del servidor local hasta que sea enviado el formulario.

Abra un nuevo documento de texto para crear la página que realizará las inserciones. La parte HTML de la misma es un formulario que contiene una tabla con los campos de texto, similar al usado en el documento `editar.php`, pero con la diferencia de que en este caso no necesitamos un campo oculto para el valor de `id`:

```

<form name="form1" method="post" action="<?echo $_SERVER['PHP_SELF'];?>">
<table width="400" border="1" cellpadding="0" cellspacing="0">
<tr>
  <td>nombre</td>
  <td><input name="nombre" type="text" id="nombre" value=""></td>
</tr>
<tr>
  <td>apellido</td>
  <td><input name="apellido" type="text" id="apellido" value=""></td>
</tr>
<tr>
  <td>nick</td>
  <td><input name="nick" type="text" id="nick" value=""></td>
</tr>
<tr>
  <td>email</td>
  <td><input name="email" type="text" id="email" value=""></td>
</tr>
<tr>
  <td>url</td>
  <td><input name="url" type="text" id="url" value=""></td>
</tr>
<tr>
  <td>&nbsp;</td>
  <td align="right"><input type="submit" name="submit" value="enviar"></td>
</tr>
</table>
</form>

```

Puede observar que hemos establecido como acción para el formulario la variable `$_SERVER['PHP_SELF']`, con lo que el formulario será enviado a esta misma página en que se declara. El botón **SUBMIT** también se declara exactamente igual que en el documento `editar.php`.

Cuando creamos la parte PHP, comprobamos primeramente la existencia o no de la variable `submit` enviada mediante POST:

```

<?php
if(isset($_POST['submit'])) {

```

Si existe, incluimos los ficheros de configuración:

```

include ("includes/config.php");
include ("includes/funciones.php");

```

Establecemos la conexión con la base de datos:

```

$cnx = conectar ();

```

Ahora realizaremos la consulta SQL para la inserción. El formato, como se explicó en el capítulo 4, es el siguiente:

```

INSERT INTO {tabla}
(campo1,...campoN)
VALUES(valor1,...ValorN);

```

Como hicimos al utilizar UPDATE para la actualización de datos, realizaremos la consulta paso a paso con el fin de familiarizarnos con la concatenación de cadenas en variables. Separaremos la consulta en tres partes, una con el nombre de las columnas que serán afectadas en la consulta (todas las de la tabla excepto la de nombre `id`), otra con la concate-

nación de las variables enviadas mediante POST, y la última, que incluirá las dos anteriores junto con la estructura para realizar la inserción.

Creamos una variable llamada `$campos` que almacena los nombres de las columnas:

```
$campos =  
"nombre,apellido,nick,email,url";
```



Nota: Observe cómo los nombres se separan entre sí utilizando una coma.

Los valores de los campos han de colocarse entre comillas simples; creamos una nueva variable de nombre `$valores` y concatenaremos en ella cada una de las variables a enviar (las disponemos de una en una para que el código resulte más legible):

```
$valores = "".$_POST['nombre']."'";  
$valores .= "".$_POST['apellido']."'";  
$valores .= "".$_POST['nick']."'";  
$valores .= "".$_POST['email']."'";  
$valores .= "".$_POST['url']."'";
```

El formato que hemos utilizado, si se fija, es el siguiente:

```
"comilla simple" + valor de variable + "comilla simple + coma"
```

Con este formato conseguiremos que el valor de las variables quede declarado entre las comillas simples y separados por una coma. El orden seguido es idéntico al especificado en la variable `$campos`, lo que es de suma importancia, ya que al ejecutar la consulta SQL se tomará el primer nombre de columna (en la variable `$campos`) junto con el primer valor a insertar en ésta (en la variable `$valores`).

Debe asegurarse además de no utilizar comas ni al final del último nombre de columna ni tras el último valor de las variables.

Formamos la consulta SQL con las dos variables que hemos creado (`$campos` y `$valores`):

```
$sql = "INSERT INTO directorio  
($campos) VALUES($valores)";
```

Después ejecutamos la consulta, especificando el error en caso de que la consulta no se realice correctamente:

```
$res = mysql_query($sql) or  
die(mysql_error());
```

Mostramos en pantalla el mensaje de que el registro fue añadido con éxito y un vínculo de vuelta al documento `listado.php`:

```
echo "Registro ingresado.<br><a  
href='listado.php'>regresar</a>";
```

Cerramos la conexión establecida con MySQL
y le ordenamos a PHP que deje de interpretar
el documento utilizando la función `exit`:

```
mysql_close($cnx);  
exit;  
?>
```

El código completo (que puede encontrar en el
documento `material/cap5/php/
nuevo.php` del CD) es el siguiente:

```
<?php  
//si la forma ha sido enviada editamos el registro.  
if(isset($_POST['submit'])) {  
  
    include ("includes/config.php");  
    include ("includes/funciones.php");  
    //nos conectamos a mysql  
    $cnx = conectar ();  
  
    $campos = "nombre,apellido,nick,email,url";  
    $valores = "'".$_POST['nombre']."'";  
    $valores .= "'".$_POST['apellido']."'";  
    $valores .= "'".$_POST['nick']."'";  
    $valores .= "'".$_POST['email']."'";  
    $valores .= "'".$_POST['url']."'";  
    $sql = "INSERT INTO directorio ($campos) VALUES ($valores)";  
    $res = mysql_query($sql) or die(mysql_error());  
    echo "Registro ingresado.<br><a href='listado.php'>regresar</a>"; mysql_close($cnx);  
    exit;  
}  
?>  
  
<form name="form1" method="post" action="<?echo $_SERVER['PHP_SELF'];?>">  
<table width="400" border="1" cellpadding="0" cellspacing="0">  
<tr>  
    <td>nombre</td>  
    <td><input name="nombre" type="text" id="nombre" value=""></td>  
</tr>  
<tr>  
    <td>apellido</td>  
    <td><input name="apellido" type="text" id="apellido" value=""></td>  
</tr>  
<tr>  
    <td>nick</td>  
    <td><input name="nick" type="text" id="nick" value=""></td>  
</tr>  
<tr>  
    <td>email</td>  
    <td><input name="email" type="text" id="email" value=""></td>  
</tr>  
<tr>  
    <td>url</td>  
    <td><input name="url" type="text" id="url" value=""></td>
```

```

</tr>
<tr>
  <td>&nbsp;</td>
  <td align="right"><input type="submit" name="submit" value="enviar"></td>
</tr>
</table>
</form>

```

Guarde el documento con este código con el nombre `nuevo.php` en la carpeta `phpmysql` de su servidor local y ábralo en su navegador tecleando `http://localhost/phpmysql/nuevo.php`, ya que para añadir un nuevo registro no necesita pasar previamente por el documento de listado. Teclee los datos para el nuevo registro y añádalo a la base de datos. Al regresar al documento `listado.php`, podrá observar cómo el registro que acaba de añadir se muestra junto con los que ya tenía almacenados, en un proceso que debe ser similar al de la figura 5.13.

Borrado de un registro específico

Si bien eliminar registros es la tarea más sencilla, es también la más delicada, ya que un solo error puede conducir a la pérdida de todos los datos almacenados. En nuestro caso no vamos a borrar el registro al invocar la página, sino que crearemos un formulario con un único botón que servirá para confirmar que desea efectuar el borrado, además de un vínculo de texto "cancelar", que utilizaremos para retornar al documento `listado.php`.

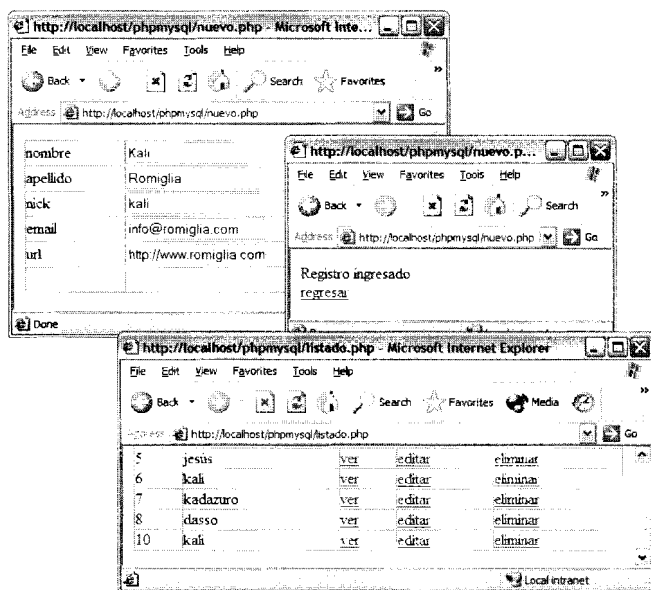


Figura 5.13.

Ingreso de registro, registro ingresado y listado al final.

El borrado se realizará utilizando como parámetro el identificador único (enviándolo mediante GET al fichero `eliminar.php`) del registro que queremos borrar. Como hicimos anteriormente, este valor se almacenará en un campo oculto del formulario y borrará el registro al ser enviado. El documento `eliminar.php` consta de tres partes diferenciadas:

- Comprobación de la existencia de la variable `id` enviada mediante GET.
- Formulario.
- Eliminación del registro.

El orden en que se dispondrá el código en la página `eliminar.php` será el siguiente:

- Eliminar registro y salir.
- Comprobar la existencia de la variable `id` enviada mediante GET.
- Formulario.

Abra un nuevo documento de texto para crear el fichero que realizará los borrados. La parte en que se elimina el registro la dejamos para el final, comenzando con la comprobación de la existencia de `id` enviada mediante GET. En caso de que no exista, se redirige al usuario al documento `listado.php`:

```
<?
//eliminar registro
//si no hay id, no puede seguir.
if(!isset($_GET['id'])){
    header("Location: listado.php");
    exit;
}
?>
```

Utilizamos el comentario `//eliminar registro` para recordar dónde debemos colocar el código PHP que realizará el borrado. A continuación de lo ya escrito, creamos la tabla y el formulario HTML:

```
<form name="form1" method="post"
action="<?echo
$_SERVER['PHP_SELF'];?>">
<table width="400" border="0"
cellpadding="0" cellspacing="0">
```

A continuación creamos la fila en la que declaramos un campo oculto con el valor de la variable `id`, junto con el texto donde se pregunta a modo de confirmación si se desea realmente borrar el registro:

```
<tr>
<td><input name="id"
type="hidden" id="id" value="<?echo
$_GET['id'];?>"><br>
    ¿Seguro de querer borrar el
registro con id <?echo
$_GET['id'];?>?</td>
</tr>
```

Observe que hemos agregado al texto de la pregunta el valor de `id` del registro para que sepa si realmente el registro seleccionado es el que usted desea borrar.

En la siguiente fila creamos un botón **SUBMIT** cuya etiqueta es "Borrar registro":

```
<tr>
<td align="right"><input
type="submit" name="submit"
value="Borrar registro."></td>
</tr>
```

La última fila de la tabla contiene un vínculo, "cancelar", utilizado para retornar al documento listado.php por si finalmente no desea efectuar el borrado del registro.

```
<tr>
  <td align="center"><a
href="listado.php">cancelar</a></td>
</tr>
```

Terminamos la parte HTML cerrando la etiqueta de la tabla y del formulario:

```
</table>
</form>
```

El fragmento de código PHP de la página que se encarga de realizar el borrado del registro (y cuyo código debe colocar donde anteriormente tecleamos la línea de comentario para ello) comienza por realizar la comprobación de la existencia de la variable submit enviada mediante el método POST por el formulario:

```
<?php
if(isset($_POST['submit'])) {
```

Si la variable existe, incluimos los ficheros de configuración:

```
include ("includes/config.php");
include ("includes/
funciones.php");
```

Establecemos la conexión con MySQL utilizando la función conectar declarada en el fichero funciones.php de la carpeta phpmysql/includes de nuestro servidor local:

```
$cnx = conectar ();
```

La consulta SQL para realizar el borrado de un registro sigue el siguiente formato:

```
DELETE FROM {tabla} WHERE campo =
valor;
```

En esta consulta, la cláusula where es la de mayor importancia, ya que es la que especifica qué registro eliminar. Éste es el lugar en el que adjuntamos el valor de la variable id enviada mediante POST cuando creamos la variable \$sql que utilizaremos para la consulta:

```
$sql = "DELETE FROM directorio WHERE
id =".$_POST['id'];
```

Efectuamos la consulta:

```
$res = mysql_query($sql) or
die(mysql_error());
```

Posteriormente, informamos del éxito en la eliminación del registro, junto con un vínculo al documento listado.php:

```
echo "Registro ".$_POST['id']."
eliminado.<br><a
href='listado.php'>regresar</a>";
```

Cerramos la conexión con MySQL y utilizamos la función exit para que PHP termine de interpretar el documento.php:

```
mysql_close($cnx);
exit;
}
```

El código completo (que puede encontrar en el documento `material/cap5/php/eliminar.php` del CD) es el siguiente:

```
<?php
if(isset($_POST['submit'])) {
    include ("includes/config.php");
    include ("includes/funciones.php");
    //nos conectamos a mysql
    $cnx = conectar ();
    $sql = "DELETE FROM directorio WHERE id=".$_POST['id'];
    $res = mysql_query($sql) or die(mysql_error());
    echo"Registro ".$_POST['id']." eliminado.<br><a href='listado.php'>regresar</a>";
    mysql_close($cnx);
    exit;
}
//si no hay id, no puede seguir.
if(!isset($_GET['id'])) {
    header("Location: listado.php");
    exit;
}
?>
<form name="form1" method="post" action="<?echo $_SERVER['PHP_SELF'];?>">
<table width="400" border="0" cellpadding="0" cellspacing="0">
<tr>
    <td><input name="id" type="hidden" id="id" value="<?echo $_GET['id'];?>"><br>
    ¿Seguro de querer borrar el registro con id <?echo $_GET['id'];?></td>
</tr>
<tr>
    <td align="right"><input type="submit" name="submit" value="Borrar
    registro."></td>
</tr>
<tr>
    <td align="center"><a href="listado.php">cancelar</a></td>
</tr>
</table>
</form>
```

Guarde el documento con el nombre `eliminar.php` en la carpeta `phpmysql` de su servidor local y abra en su navegador el documento de listado tecleando `http://localhost/phpmysql/listado.php`. Cuando seleccione un registro para eliminar, será enviado al documento `eliminar.php` que acaba de realizar, en el que se le pedirá confirmación para eliminar dicho registro.

Cuando pulse el botón **SUBMIT** con el texto "Borrar registro", se mostrará en pantalla el mensaje de que se realizó el borrado con éxito y un enlace al documento `listado.php`, que mostrará, cuando acceda a él, todos los registros a excepción del que acabamos de eliminar.

```
gotoAndPlay("capitulo_6");
```

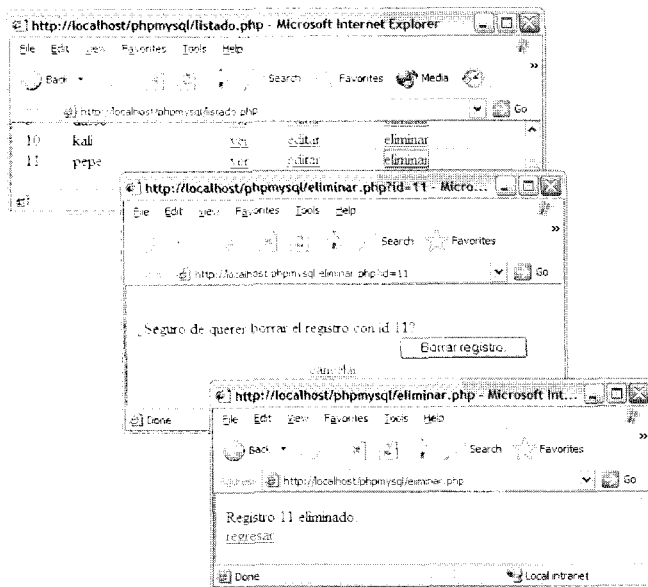


Figura 5.14.

Listado, eliminación de registro y mensaje de confirmación de borrado.

Capítulo 6

Flash, PHP y MySQL

Los anteriores cinco capítulos nos han servido como repaso de las nociones básicas del funcionamiento tanto de Flash (y sus conexiones con contenidos dinámicos) como de PHP y MySQL. Este capítulo, cuyos contenidos son la razón de ser de este libro, puede considerarse como el punto en el que confluye todo lo explicado hasta el momento: trabajaremos con documentos realizados en Flash que obtendrán la información que muestren de la base de datos de que disponemos en el servidor local y que hemos utilizado para realizar los ejemplos de los capítulos anteriores.

Trabajar conjuntamente con Flash y PHP/MySQL dependerá, en última instancia, de la versión de Flash de que dispongamos en nuestro ordenador, pero lo realmente importante es que sepamos qué es lo que estamos haciendo. Si tenemos esto claro, el resto es

simplemente adaptarse a la sintaxis necesaria para cada versión.

Viene este último comentario a colación de que en las versiones anteriores a Flash MX teníamos que hacer uso de las acciones `loadVariables` y `loadVariablesNum` para acceder a contenidos de texto dinámicos. Su utilización sigue siendo perfectamente válida en las nuevas versiones del programa, pero en éstas disponemos, tal y como vimos en el capítulo 1, del objeto predeterminado *LoadVars*, cuyo cometido es, precisamente, almacenar en su interior la información a la que accede, a diferencia de lo que hacen `loadVariables` y `loadVariablesNum`, que almacenan la información en el clip de película o el nivel que se especifique al llamarlas.

Primeros pasos

La idea principal a la hora de utilizar Flash conjuntamente con documentos PHP es la siguiente:

- Flash realiza una llamada al documento PHP.
- Opcionalmente, PHP se conecta a la base de datos y extrae resultados de una consulta.
- PHP escribe los resultados de su acción utilizando un "formato" para los datos que Flash pueda entender.

Cuando hablamos de utilizar un "formato" que Flash entienda, nos referimos a la forma en que PHP escribe sus resultados. Si recuerda, en el capítulo anterior, cada vez que un documento PHP mostraba la información extraída de la base de datos, utilizaba un formato HTML para tal tarea, como por ejemplo en el siguiente código:

```
if(mysql_num_rows($res) >0){
    //impresión de los datos.
    while (list($id,$nick) = mysql_fetch_array($res)) {
        echo "<tr><td>$id</td>\n";
        echo "<td>$nick</td>\n";
        echo "<td><a href='ver.php?id=$id'>ver</a></td>\n";
        echo "<td><a href='editar.php?id=$id'>editar</a></td>\n";
        echo "<td><a href='eliminar.php?id=$id'>eliminar</a></td></tr>\n";
    }
}else{
    echo "<td colspan='5' align='center' >no se obtuvieron resultados</td>";
}
```

En cambio, cuando PHP va a enviar información a Flash, no va a escribir los resultados de sus acciones en este formato, sino que ha de hacerlo de forma que Flash pueda procesar los datos recibidos, por lo que éstos deben ser escritos en un formato que entiendan los objetos predeterminados *LoadVars* y XML de Flash MX.

Cómo utilizar el material de este capítulo

Tras realizar todas las consultas relacionadas a lo largo del capítulo 5 nos encontramos en este momento con que el contenido de la tabla *directorio* de la base de datos *pruebas* que estamos utilizando posee unos contenidos como los que se muestran en la figura 6.1.

pruebas.directorio ejecutándose en localhost - phpMyAdmin 2.5.1 - Microsoft Internet Explorer

Base de datos pruebas - Tabla directorio ejecutándose en localhost

Mostrando registros 0 - 0 (0 total, La consulta tardó 0.0092 seg)

consulta SQL: [Editar] [Exportar el SQL] [Copiar código PHP]
 SELECT
 FROM `direc` LIMIT 0, 25

Mostrar: 30 filas empezando de 0 en modo horizontal y repite encabezados cada 100 celdas

	id	nombre	apellido	email	url	nick
Editar Borrar	1	Rodolfo	Ruiz	rol@alesys.net	http://www.alesys.net	rolf
Editar Borrar	2	Daniel	de la Cruz	nula@granata.com	http://www.granata.com	granata
Editar Borrar	3	Fernando	Flórez	info@onelt.com	http://www.onelt.com	fernando
Editar Borrar	4	Santiago	Anglés	info@sangles.com	http://www.sangles.com	sangles
Editar Borrar	5	Manuel	Vejarano	xflavio@ashmail.com	http://elflash.tk.com	jesus
Editar Borrar	6	Kali	Romiglia	kali@romiglia.com	http://www.romiglia.com	kali
Editar Borrar	7	Carlos	Zumbado	nadiel@kadazuro.com	http://www.kadazuro.com	kadazuro
Editar Borrar	8	Ricardo	Salazar	info@nomaster.com	http://www.nomaster.com	dasso
Editar Borrar	10	Kali	Romiglia	kali@romiglia.com	http://www.romiglia.com	kali

Figura 6.1.

Contenidos de la tabla directorio de la base de datos pruebas.

Por otra parte, y para no mezclar los nuevos contenidos con los que trabajamos en los capítulos anteriores, cree una nueva carpeta con el nombre `phpflash` en la raíz de su servidor local. Esta carpeta será la que utilizaremos para realizar los ejercicios de este capítulo, que puede encontrar en la carpeta `material/cap6` del CD. Una vez creada, el método que seguiremos será el de indicar qué ficheros del CD ha de copiar a dicha carpeta, abrirlos en su navegador para ver los resultados de sus acciones y posteriormente editarlos (los contenidos en la carpeta `phpflash`) para explicar cómo y por qué se han implementado de esa forma.

Obtención de datos usando el objeto *LoadVars*

En el capítulo 1 del libro analizamos cómo obtener información almacenada en documentos de texto `.txt` utilizando el objeto predeterminado *LoadVars* de Flash MX. Si recuerda, el modo en que debía guardarse la información en dichos documentos consistía en especificar las distintas variables mediante el carácter `&` utilizando el siguiente formato:

```
&variable1=valor1&variable2=valor2&variable3=valor3
```

Una vez que el objeto *LoadVars* cargaba correctamente los contenidos del documento `.txt`, podíamos acceder a los valores de las variables utilizando la sintaxis:

```
objeto_LoadVars.variable1
```

En este ejemplo, el valor almacenado en `objeto_LoadVars.variable1` es "valor1". Por tanto, cuando deseemos almacenar en un objeto *LoadVars* información externa

generada por un documento PHP, hemos de procurar que éste la escriba en el mismo formato que utilizamos para los documentos de texto .txt.

Ejemplo básico

Veamos un ejemplo muy básico de cómo PHP escribe datos en el formato que interpreta *LoadVars*. Vaya a la carpeta `material/cap6/loadvars/basico` del CD y copie los documentos que encontrará allí en la carpeta `phpflash` de su servidor local.

Edite el documento `una_variable.PHP`, en el que encontrará este pequeño fragmento de código:

```
<?php
$valor=35;
echo "&valor=".$valor;
?>
```

Este PHP simplemente escribe el contenido de la variable `$valor` precedido de la cadena `"&valor="`. Abra ahora el documento en su navegador tecleando `http://localhost/phpflash/una_variable.php`, con lo que obtendrá en su pantalla es el siguiente texto (tal y como puede ver en la figura 6.2):

```
&valor=35
```

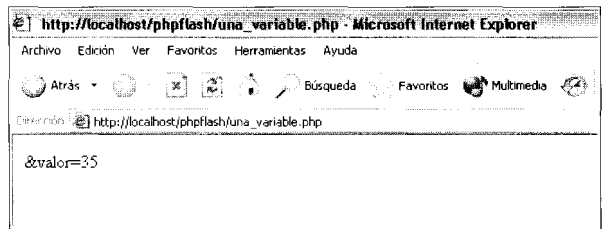


Figura 6.2.

*Texto generado por el documento
una_variable.php.*

En vez de escribir dentro de una tabla HTML el valor de la variable, como hacíamos en el capítulo anterior, utilizamos el formato que explicamos previamente para crear los documentos .txt que almacenaban información. Así, cuando el objeto *LoadVars* realice la llamada al documento `una_variable.php`, éste generará el texto que hemos visto en el párrafo anterior, de modo que, a efectos prácticos, el resultado que obtiene el objeto *LoadVars* al llamar a este documento PHP, es el mismo que si hubiese llamado a un documento de nombre, pongamos por caso, `una_variable.txt`, cuyo contenido fuese:

```
&valor=35
```

Abra ahora el documento `basico fla`, cuyo fotograma 1 de la capa *acciones* contiene el siguiente código:

```
datos=new LoadVars();
```

Para comenzar, creamos un objeto *LoadVars* que será el que almacene los contenidos externos. Posteriormente definimos el evento `onLoad` para dicho objeto, cuyas acciones se ejecutarán cuando se produzca la carga, exitosa o no, de dichos contenidos:

```
datos.onLoad=function(exito){
    if (exito){
```

En caso de que los datos hayan sido accedidos correctamente y almacenados, por tanto, dentro del objeto en forma de variables, los recorreremos utilizando un bucle del tipo `for...in` para mostrarlos en la ventana de Salida:

```
for (i in this){
    trace(i+": "+this[i]);
    trace("_____");
}
} else {
```

Si se ha producido algún error en el acceso a los datos, se muestra un mensaje en la ventana de Salida informando de tal circunstancia:

```
    trace("Error al cargar los datos");
}
}
```

Finalmente, escribimos la acción para que el objeto *LoadVars* realice la llamada al documento PHP:

```
datos.load("http://localhost/
phpflash/una_variable.php");
```

Si ahora reproduce su película (Menú **Control>Probar película**, o bien pulsando **Control-Intro**), obtendrá la siguiente información en su ventana de Salida (figura 6.3):

```
valor: 35
-----
onLoad: [type Function]
-----
```

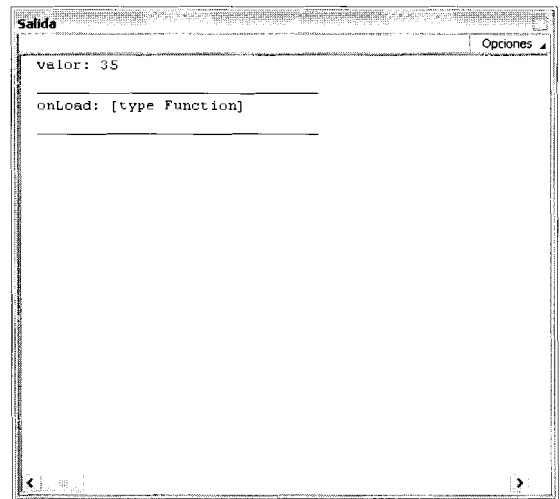


Figura 6.3.

Contenido de la ventana de Salida al reproducir la película.

Si reemplaza en el documento .fla el bucle `for...in` por la línea (como se ha hecho en el documento *basico2.fla*):

```
trace(datos.valor);
```

de forma que el código sea el siguiente:

```
datos=new LoadVars();

datos.onLoad=function(exito){
    if (exito){
        trace(datos.valor)
    } else {
        trace("Error al cargar los
datos");
    }
}

datos.load("http://localhost/
phpflash/una_variable.php");
```

Al reproducir su película, obtendrá en su ventana de Salida lo siguiente:

35

que es justamente el valor que contiene la variable `valor`. Como puede ver, es relativamente sencillo obtener los datos generados por un documento PHP. En el siguiente apartado utilizaremos, además de documentos PHP, la base de datos `pruebas` de la que disponemos en nuestro servidor local.

Obtención del listado de usuarios

En este apartado vamos a crear un documento PHP que se conecte a la base de datos `pruebas` y obtenga el `nick` y el identificador único `id` de cada uno de los registros de la tabla `directorio`, para posteriormente escribirlos en el formato que usa `LoadVars`:

```
&nick1=rolf&id1=1&nick2=granatta&id2=2...
```

Además de dichos datos, el documento PHP también escribirá algunas variables más dependiendo de si obtiene los datos con éxito o, por el contrario, se produce algún error. En el primero de los casos se crean una variable de nombre `output` (en la que se almacena el texto "ok") y otra variable de nombre `total` (que almacena el número total de usuarios contenidos en la tabla). Si, por el contrario, se produce algún error, se crea igualmente la variable `output` (aunque almacenando el texto "error"), pero en vez de una variable de nombre `total`, se crea una de nombre `msg` (en la que se almacena el mensaje de error que se ha producido). Por último, si nuestra consulta sobre la tabla es correcta pero no obtiene resultados, se crean las variables `output` (almacenando la cadena "error") y `msg` (almacenando la cadena "No hay datos").

Elimine los documentos de la carpeta `phpflash` de su servidor local y copie allí los que encontrará en la carpeta `material/cap6/loadvars/carga_simple` del CD. Edite el documento `loadVars_ids.php`, en el que encontrará el siguiente código:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");
Incluimos los dos ficheros de la
carpeta includes en los que hemos
declarado las variables y funciones
de uso común. Posteriormente nos
conectamos a MySQL:
$cnx = conectar();
```

Definimos una variable de nombre `$sql` con la consulta que queremos realizar (la selección de los contenidos de `nick` e `id` de todos los registros de la tabla `directorio`) y la efectuamos almacenando el resultado en la variable `$res`:

```
$sql = "SELECT id,nick FROM
directorio";
$res = mysql_query($sql) or
die("output=error&msg=".mysql_error());
```

A continuación comprobamos si existe más de una fila en la variable que almacena el resultado. De ser así, creamos una variable llamada `$salida` en la que vamos a ir escribiendo los datos resultantes de la consulta, comenzando por establecer la variable `output` y su valor "ok":

```
if(mysql_num_rows($res) > 0){
    $salida = "&output=ok&";
    $contador = 0;
```

Utilizamos un bucle `while` para obtener la información (añadiéndola al contenido de la variable `$salida`), incrementando en una unidad el contenido de la variable `$contador` para la siguiente iteración:

```
while(list($id,$nick)= mysql_fetch_array($res)){
    $contador++;
    $salida.="id$contador=$id&nick$contador=".utf8_encode($nick)."&";
}
```

La función `utf8_encode` de PHP, cuya sintaxis es:

```
utf8_encode(cadena de texto);
```

se utiliza para dar formato a los acentos y/o caracteres especiales con la finalidad de que Flash los interprete correctamente. Si se especifica el formato *Unicode* para el texto, Flash lo interpreta como *Unicode*. Si no se especifica formato alguno, Flash lo interpreta como UTF-8. La razón es la de que la versión 6 del *reproductor de Flash* analiza los primeros dos bytes del archivo buscando una marca (*Byte Order Mark, BOM*), una convención utilizada para identificar el formato *Unicode*. Si no se encuentra ninguna marca, el texto se interpreta como si hubiera sido codificado mediante el formato UTF-8 (un formato de codificación de 8 bits).



Nota: El fichero en sí no se imprime con formato UTF-8, sino sólo aquellos campos que son susceptibles de contener acentos y/o caracteres especiales, para que Flash, al no detectar *Unicode*, interprete todos los contenidos como UTF-8.

Una vez que salimos del bucle `while`, sabemos que en la variable `$contador` se encuentra el número total de registros de la tabla, por lo que utilizamos ese valor para asignárselo a una nueva variable llamada `total`. Para finalizar, imprimimos el contenido de la variable `$salida`, liberamos memoria y cerramos la conexión con la base de datos:

```
$salida.="total=$contador&";
echo $salida;
mysql_free_result($res);
mysql_close($cnx);
}else{
```

En caso de que no existan datos en la tabla establecemos como "error" el contenido de la variable `output` y "No hay datos" como contenido de una nueva variable llamada `msg`:

```
echo "output=error&msg=No hay
datos";
}
?>
```

El código completo del documento

loadVars_ids.php es el siguiente:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");

//nos conectamos a mysql.
$cnx = conectar();
// consulta sql.
$sql = "SELECT id,nick FROM directorio";
//ejecutamos la consulta sql
$res = mysql_query($sql) or die("output=error&msg=".mysql_error());
//contamos el número de filas en el resultado.
if(mysql_num_rows($res) > 0){
    //si hay datos.
    $salida = "&output=ok&";
    $contador = 0;
    //parseamos la información guardándola en $salida.
    while(list($id,$nick)= mysql_fetch_array($res)){
        //aumentamos en 1 el contador
        $contador++;
        $salida.="id$contador=$id&nick$contador=".utf8_encode($nick)."&";
    }
    //agregamos el total de registros a la salida.
    $salida.="total=$contador&";
    //imprimimos la salida.
    echo $salida;
    //liberamos memoria
    mysql_free_result($res);
    //cerramos la conexión
    mysql_close($cnx);
}
else{
    //no hay datos, pasamos el mensaje a flash.
    echo "output=error&msg=No hay datos";
}
?>
```

Si ahora abre este documento en su navegador

tecleando <http://localhost/phpflash/>

loadVars_ids.php, obtendrá como resultado la

siguiente cadena de texto:

```
&output=ok&id1=1&nick1=rolf&id2=2&nick2=granatta&id3=3&nick3=fernando&id4=
4&nick4=sangles&id5=5&nick5=jesÃ°s&id6=6&nick6=kali&id7=7&nick7=kadazuro&id8=8&nick8=
dasso&id9=10&nick9=kali&total=9&
```

Puesto que disponemos de nueve usuarios en nuestra tabla *directorio*, la variable *total* almacena el valor 9 y, como se ha realizado correctamente la consulta, *output* almacena

la cadena "ok". Además, se han generado los pares de *nick* e *id* para cada uno de los usuarios de la tabla en el formato con el que estamos trabajando en este apartado.

Ahora abra el documento

carga_simple fla y vaya al fotograma 1 de la capa *acciones*, en el que encontrará el siguiente código:

```
datos_bd=new LoadVars();
```

Creamos un nuevo objeto *LoadVars* y definimos su evento *onLoad* para cuando se produzca la carga de los datos:

```
datos_bd.onLoad=function(exito){  
    if (exito){
```

Se definen las acciones si la carga se ha realizado con éxito, en cuyo caso pueden ocurrir dos cosas dependiendo de si el valor almacenado en la variable *output* es "ok" o bien "error":

```
        if (this.output=="ok"){
```

Si el valor contenido en *output* es "ok", entonces se habrán creado todos los pares de variables referentes a los usuarios, y las listamos utilizando un bucle del tipo *for...in*:

```
            for (i in this){  
                trace(i+": "+this[i]);  
                trace("-----");  
            }  
        } else {
```

Si el valor contenido en *output* es "error", se muestra un mensaje de error, especificado por PHP en la variable de nombre *msg*:

```
            trace("Se produjo el siguiente error: "+this.msg);  
        }  
    } else {
```

Si se produce un error al acceder a los datos, se avisa de tal circunstancia:

```
        trace("Error al cargar los datos");  
    }  
}
```

Finalmente, se realiza la llamada del objeto *LoadVars* al documento PHP:

```
datos_bd.load("http://localhost/  
phpflash/loadVars_ids.php");
```

Si ahora reproduce la película, obtendrá la siguiente información en su ventana de Salida:

```
total: 9  
-----  
nick9: kali  
-----  
id9: 10  
-----  
nick8: dasso  
-----  
id8: 8  
-----  
nick7: kadazuro  
-----  
id7: 7  
-----  
nick6: kali  
-----  
id6: 6  
-----
```

```

nick5:  jesús
-----
id5:  5
-----
nick4:  sangles
-----
id4:  4
-----
nick3:  fernando
-----
id3:  3
-----
nick2:  granatta
-----
id2:  2
-----
nick1:  rolf
-----
id1:  1
-----
output:  ok
-----
onLoad:  [type  Function]
-----

```

No obstante, este listado de variables y sus respectivos valores es algo relativamente abstracto, puesto que el objetivo último de acceder a la información es la de utilizarla para presentarla en pantalla de forma visible, lo que vamos a realizar en el siguiente apartado de este capítulo: el documento .fla recibirá del documento .php el listado conteniendo los campos `nick` e `id` de los usuarios, y cada uno de estos últimos utilizará dicha variable `id` para acceder a sus detalles.

Utilización del objeto LoadVars y dos documentos PHP

Elimine antes de comenzar los documentos de la carpeta `phpflash` de su servidor local y copie allí los que encontrará en la carpeta `material/cap6/loadvars/` `carga_separada` del CD. Abra en su navegador el documento `carga_separada.html` tecleando `http://localhost/phpflash/`

`carga_separada.html`; el contenido que verá ha de ser un listado de *nicks* de los usuarios de la tabla ordenados de forma ascendente por su identificador `id`. Para conseguir esta información hemos utilizado el documento `loadVars_ids.php`, que es exactamente igual al del apartado anterior. Su función, por tanto, es la misma: escribir los contenidos de los campos `nick` e `id` de cada uno de los registros de la tabla. Flash recoge esos contenidos y los mostrará en la forma que puede verse en la figura 6.4.

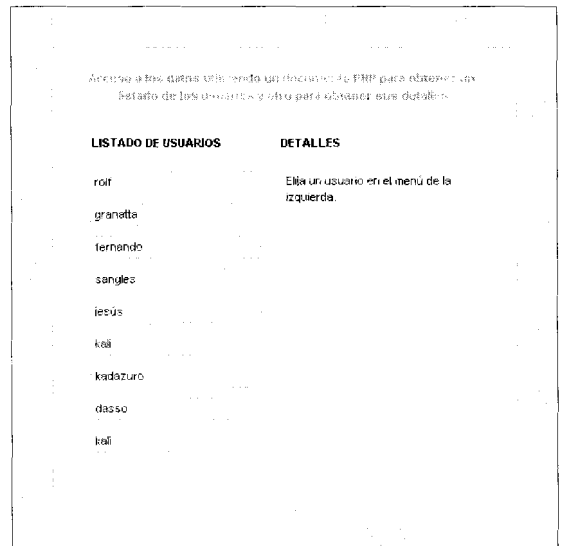


Figura 6.4.

Listado de usuarios tras la consulta a la base de datos.

Si ahora pulsamos sobre uno de los usuarios, por ejemplo, *sangles*, obtendremos una información más detallada en la columna derecha de la pantalla. Al pulsar sobre un usuario, Flash envía su identificador `id` mediante el método `POST` al otro documento PHP de que disponemos, `loadVars_datos.php`. La función de este documento es la de realizar

una consulta a la base de datos basándose en la variable `id` cuyo valor se recibe mediante POST, escribiendo entonces el contenido de los campos `nombre`, `apellido`, `email` y `url` del registro de la tabla cuyo identificador único coincide con el valor de la variable `id`. Flash recoge esos datos y los mostrará, como puede verse en la figura 6.5.

Atención: los datos están en un directorio "lib" para sistemas no administrados por Moodle y sólo para obtener los detalles

LISTADO DE USUARIOS	DETALLES
rofi	id: 4
granalla	nombre: Santiago Anglés
fernando	nick: sangles
sangles	email: info@sangles.com
jesus	url: http://www.sangles.com
kali	
ladazuro	
dasso	
kali	

Figura 6.5.

Detalles del usuario sangles.

Edite este documento

(`loadVars_datos.php`), en el que encontrará el siguiente código:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");
```

Comenzamos, igual que siempre, "incluyendo" los documentos de la carpeta `includes` en los que se almacenan las variables y funciones de uso común. Nos conectamos a la base de datos MySQL y almacenamos en la variable `$sql` la consulta que queremos realizar; en este caso, seleccionar todos los

campos del registro cuyo identificador único (`id`) coincida con el valor de la variable `id` enviada mediante POST:

```
$cnx = conectar();
$sql = "SELECT * FROM directorio
WHERE id=".$_POST['id'];
```

Ejecutamos la consulta SQL y almacenamos el resultado de la misma en la variable `$res` o imprimimos el mensaje de error generado si se producen problemas:

```
$res = mysql_query($sql) or
die("output=error&msg=".mysql_error());
```

Después, contamos el número de filas que ha almacenado dicha variable para saber si hay o no registros que cumplan con los requisitos de la consulta.

```
if(mysql_num_rows($res) > 0){
```

En caso de que haya algún registro cuyo `id` coincida con el valor de `id` enviado mediante POST, generamos, igual que antes, una variable `output` de contenido "ok", y posteriormente recorremos la información guardándola en la variable `$salida` (observe cómo sólo se formatean con UTF-8 los campos susceptibles de contener acentos y/o caracteres especiales: `nombre`, `apellido` y `nick`):

```

$salida = "&output=ok&";
while($fila = mysql_fetch_array($res)){
    $salida.="id=".$fila['id'];
    $salida.="&nombre=".utf8_encode($fila['nombre']);
    $salida.="&apellido=".utf8_encode($fila['apellido']);
    $salida.="&nick=".utf8_encode($fila['nick']);
    $salida.="&email=".$fila['email'];
    $salida.="&url=".$fila['url']."&";
}

```

Al terminar la ejecución del bucle `while` se imprime el contenido de la variable `$salida`, se libera la memoria y se cierra la conexión con la base de datos:

```

echo $salida;
mysql_free_result($res);
mysql_close($cnx);
}else{

```

Si llegamos a este punto, no hay ningún registro de la tabla que cumpla con los requisitos que se especificaban en la consulta SQL, de modo que se escriben una variable `output` (cuyo contenido es "error") y otra llamada `msg` (cuyo contenido es "No hay datos"):

```

echo "output=error&msg=No hay datos";
}
?>

```

Por tanto, cuando pulse sobre el usuario *sangles*, la cadena de texto que va a generar este documento PHP, si la consulta se efectúa correctamente, es:

```

&output=ok&id=4&nombre=Santiago&apellido=Anglés&nick=sangles&email=info@sangles.com&url=http://www.sangles.com&

```

El código completo del documento `loadVars_datos.php` es el siguiente:

```

<?php
include ("includes/config.php");
include ("includes/funciones.php");

//nos conectamos a mysql.
$cnx = conectar();

```

```
// consulta sql.
$sql = "SELECT * FROM directorio WHERE id=".$$_POST['id'];
//ejecutamos la consulta sql
$res = mysql_query($sql) or die("output=error&msg=".mysql_error());
//contamos el número de filas en el resultado.
if(mysql_num_rows($res) > 0){
    //si hay datos.
    $salida = "&output=ok&";
    //parseamos la información guardándola en $salida.
    while($fila = mysql_fetch_array($res)){
        $salida.="id=".$fila['id'];
        $salida.="&nombre=".utf8_encode($fila['nombre']);
        $salida.="&apellido=".utf8_encode($fila['apellido']);
        $salida.="&nick=".utf8_encode($fila['nick']);
        $salida.="&email=".$fila['email'];
        $salida.="&url=".$fila['url']."&";
    }
    //imprimimos la salida.
    echo $salida;
    //liberamos memoria
    mysql_free_result($res);
    //cerramos la conexión
    mysql_close($cnx);
}else{
    //no hay datos, pasamos el mensaje a flash.
    echo "output=error&msg=No hay datos";
}
?>
```

Abra ahora el documento `carga_separada fla`, en cuyo primer fotograma encontrará, en la capa *textos*, la distribución de los contenidos que pueden verse en la pantalla, así como un campo de texto con formato HTML cuyo nombre de instancia es *detalles*. Si abre la Biblioteca (pulsar **F11**, o bien Menú Ventana>Biblioteca), encontrará un clip de película llamado *nombre*, al que hemos asignado *nombre* como nombre de vinculación de la misma forma que explicamos en el capítulo 1. Además, el interior de este clip de película contiene un campo de texto cuyo nombre de instancia es *nick*.

Lo que vamos a hacer en este documento Flash es crear un objeto *LoadVars* que realizará una llamada al documento `loadVars_ids.php` para obtener el listado de *nick* e *id* de los

usuarios, además de la variable *total* en la que está almacenado el número de usuarios de la tabla. Utilizaremos esta última para crear un bucle *for* que importe una instancia del clip *nombre* de la Biblioteca una vez por cada iteración, y asignaremos a dicho clip un valor de *id* y un *nick*, que será mostrado en el campo de texto que posee el clip.

Posteriormente, definiremos las acciones que ha de efectuar cada clip cuando pulsemos sobre él, que básicamente consisten en crear otro objeto *LoadVars* que llamará al documento `loadVars_juntos.php`, enviando mediante POST el valor del *id* de ese usuario. Si el documento PHP recibe con éxito los datos de la consulta, los enviará de nuevo al objeto *LoadVars* que le llamó en primera instancia, de forma que, una vez recibida, la información se

procesa y formatea para ser mostrada en el campo de texto de la derecha.

El código para realizar estas tareas puede encontrarlo en el fotograma 1 de la capa *acciones*:

```
// declaración de variables a
utilizar
inicioX=45;
inicioY=125;
separacionVertical=30;
```

Comenzamos declarando unas variables que utilizaremos posteriormente: *inicioX* e *inicioY*. Estos dos valores definen las coordenadas del punto a partir del cual comenza-

remos a colocar las instancias del clip *nombre* que importemos en nuestra película, separadas verticalmente unas de otras una distancia cuyo valor se especifica en la variable *separacionVertical*.

La siguiente función va asociada al evento *onLoad* de un objeto *LoadVars*; por tanto, cuando hacemos referencia a *this*, estamos referenciando el propio objeto. Cuando esta función se ejecute será porque el objeto *LoadVars* ha recibido los datos de un usuario tras haber enviado un valor de *id*, y mostraremos esos datos en el campo de texto *detalles*, dando formato HTML al texto para que los enlaces al correo y al sitio Web sean pulsables:

```
// función que mostrará los detalles de cada usuario en el campo de texto de la
derecha
mostrarDetalle=function(exito){
    if (exito){
        if (this.output=="ok"){
            detalles.htmlText="";
            detalles.htmlText+="id: </b>"+this.id+"<br>";
            detalles.htmlText+="nombre: </b>"+this.nombre + "
"+this.apellido+"<br>";
            detalles.htmlText+="nick: </b>"+this.nick+"<br>";
            detalles.htmlText+="email: </b><u><a
href=\"mailto:\"+this.email+\"\">"+this.email+"</a></u><br>";
            detalles.htmlText+="url: </b><u><a href=\"\"+this.url+\"\"
target=\"_blank\">"+this.url+"</a></u><br>";
        } else {
            detalles.htmlText="Se produjo el siguiente error: <b>"+this.msg+"</
b>";
        }
    } else {
        detalles.htmlText="Error al cargar los datos del usuario";
    }
}
```

La siguiente función se ejecuta cuando se pulsa sobre uno de los usuarios, razón por la que se pasa el identificador *id* de ese usuario como parámetro de la función. El cometido de ésta última consiste en crear un nuevo objeto *LoadVars* (local de la función) y crear un atributo para el mismo de nombre *id*. A ese atri-

buto se le asigna el valor pasado por parámetro y posteriormente se realiza una llamada al método `sendAndLoad` del objeto *LoadVars* cuya sintaxis es la siguiente:

```
objeto_LoadVars.sendAndLoad(ruta_fichero, objeto_receptor, método_envío);
```

donde `ruta_fichero` es la ruta en el servidor al, en este caso, documento PHP que realizará las acciones, `objeto_receptor` es el objeto en el que se recibirán las variables que se obtengan de la acción del fichero especificado en `ruta_fichero`, y por último, `método_envío` sirve para especificar POST y/o GET como método para enviar las variables.

Por tanto, la función `obtenerDetalle` crea un objeto *LoadVars* y un atributo para el mismo al que asigna un valor. Utilizando `sendAndLoad` se envía al documento PHP ese

atributo que se ha creado (`send`, aunque si hubiéramos creado varios se enviarían todos ellos). El documento PHP, recibiendo ese dato, escribirá una respuesta en formato de variables especificadas mediante caracteres `&`, y sus correspondientes valores que se enviarán (`load`) al objeto indicado en la llamada del método `sendAndLoad`, en este caso, el mismo objeto que la realizó. Como siempre, declaramos una función que se asocie al evento `onLoad` del objeto, para que se procesen los datos una vez hayan sido recibidos y almacenados dentro del mismo:

```
// función que extrae los detalles del usuario cuyo id se pasa por parámetro
obtenerDetalle=function(id){
    var detalle_usuario=new LoadVars();
    detalle_usuario.id=id;
    detalle_usuario.onLoad=mostrarDetalle;
    detalle_usuario.sendAndLoad("http://localhost/phpflash/loadVars_datos.php",detalle_usuario,"POST");
}

// objeto LoadVars que recibe el listado de usuarios
datos_bd=new LoadVars();
```

La línea anterior crea un nuevo objeto *LoadVars*, y a continuación especificamos las acciones de su evento `onLoad`, que se ejecutarán cuando se carguen los datos en el objeto:

```
// función que procesa la información del listado de usuarios
datos_bd.onLoad=function(exito){
    if(exito){
```

En caso de que el valor almacenado en `exit` sea `true` (es decir, se hayan cargado correctamente los contenidos generados por el documento PHP), se comprueba si el valor de la variable `output` es "ok" o "error". Si se da este último caso, se muestra en el campo de texto `detalles` el mensaje de error generado por PHP. En caso de que todo sea correcto, primeramente se convierte a número el contenido almacenado en la variable `total` (que es una cadena de texto) utilizando la función `parseInt`, cuya sintaxis es la siguiente:

```
parseInt(cadena_texto);
```

y cuya tarea es la de convertir una cadena de texto en número siempre que sea posible. Por ejemplo:

```
parseInt("16");
```

retornará el número 16. En cambio:

```
parseInt("hola");
```

retornará el valor NaN (Not A Number). Una vez que hemos convertido en número el

contenido de la variable, lo utilizamos para crear un bucle de tipo `for` cuyo índice comienza con el valor 1 y termina con el valor especificado en `total`, incrementándose una unidad por cada iteración. En cada una de éstas se importa una instancia del clip *nombre* de la Biblioteca y se le asigna como nombre de instancia la cadena de texto resultante de concatenar "nombre" con el valor actual en que se encuentre el índice del bucle. A este nuevo clip se le asigna un nivel de profundidad indicado por el índice del bucle incrementado en una unidad, y como último parámetro vamos a crear un objeto en el que se especificará la posición horizontal y vertical del clip en el escenario. Observe cómo se asigna el resultado de la llamada al método `attachMovie` del objeto *MovieClip* a una variable llamada `nom`. Realizando la llamada de esta forma, posteriormente puede usted referenciar al clip recién creado por `nom`, una alternativa al modo en que lo referenciamos en el capítulo 1 siguiendo la sintaxis `_root["nombre"+n]`:

```
if(this.output=="ok"){
    this.total=parseInt(this.total);
    for(var n=1;n<this.total+1;n++){
        var nom=_root.attachMovie("nombre","nombre"+n,n+1,
        {_x:inicioX,_y:inicioY+(separacionVertical*(n-1))});
    }
```

La posición en X será siempre la misma y está definida por el valor de la variable `inicioX`. En cambio, el valor de la posición en Y varía en cada iteración del bucle, de manera que utilizamos el valor del índice para especificar la posición vertical de acuerdo a la expresión:

```
inicioY+(separacionVertical*(n-1))
```

Los valores de posición en Y que se generan para las instancias del clip de película pueden verse en la tabla 6.1:

Tabla 6.1.

Valores de colocación en el eje Y de las instancias de clips importadas.

Valor de n	Nombre de la instancia de clip importada	Cálculos para posicionar la instancia verticalmente	Posición en el eje vertical (_y)
1	_root.nombre1	125+(30*0)	125
2	_root.nombre2	125+(30*1)	155
3	_root.nombre3	125+(30*2)	185
...

Una vez que hemos colocado la instancia del clip de película, hemos de rellenar el campo de texto `nick` con el nombre correspondiente, para lo que usaremos de nuevo el índice del bucle; concatenándolo con la cadena de texto `nick` obtendremos el nombre de variable (`nick1`, `nick2`, `nick3`, ...) para nuestra instancia de clip recién creada. El valor de la variable `id` que se crea en el clip de película se

asigna de igual forma, pero concatenando el índice del bucle con la cadena de texto `id`.

```
nom.nick.text=this["nick"+n];
nom.id=this["id"+n];
```

Y para terminar con la instancia del clip, definimos las acciones para su evento `onRelease`, que consistirán en invocar a la función `obtenerDetalle` utilizando como parámetro el valor de `id` almacenado en la instancia de clip.

```
nom.onRelease = function (){
    this._parent.obtenerDetalle(this.id);
}
detalles.htmlText="Elija un usuario en el menú de la izquierda.";
```

También creamos, mediante la línea anterior, un pequeño mensaje de "bienvenida" para el usuario que haya accedido al documento .html, indicándole lo que ha de hacer para obtener los detalles de los usuarios.

```
    }else{
        detalles.htmlText="Se produjo el siguiente error: <b>"+this.msg+"</b>";
    }
}
}else{
```

Si se llega a este punto es porque se ha producido un error que ha impedido que el objeto *LoadVars* importe los datos que el documento PHP ha de escribir.

```
    detalles.htmlText="<b>Error al cargar los datos</b>";
}
}
```

Finalmente, llamamos al documento `loadVars_ids.php` para obtener el primer listado de usuarios:

```
datos_bd.load("http://localhost/phpflash/loadVars_ids.php");
```

El orden en que se ejecutan estas acciones no es el mismo en que se han declarado. Una secuencia de funcionamiento de este código podría ser el siguiente:

- El documento .fla crea un objeto *LoadVars* de nombre `datos_bd`.
- `datos_bd` realiza una llamada al documento PHP `loadVars_ids.php`.
- `loadVars_ids.php` se conecta a la base de datos para obtener todos los `nick` e `id` de los registros almacenados para posteriormente escribirlos en el formato `&variable=valor`.

- `datos_bd` recibe correctamente los datos escritos por el documento PHP y se ejecutan las acciones de `datos_bd.onLoad`, que importan tantas veces una instancia del clip *nombre* como usuarios haya en la tabla (adjudicando a cada clip una variable con el `id` y rellenando el campo de texto con el `nick`).

Cuando pulsamos sobre uno de los clips, por ejemplo, tal y como vimos anteriormente, el del usuario *sangles*:

- El clip que muestra el `nick sangles`, cuyo `id` es 4, llama a la función `obtenerDetalle` pasando el valor 4 como parámetro.
- `obtenerDetalle` crea un objeto *LoadVars* llamado `detalle_usuario` y le

asigna el valor pasado por parámetro (4) a un nuevo atributo al que llama `id`.

- `detalle_usuario`, usando `sendAndLoad`, envía este valor (4) al documento `loadVars_datos.php` mediante el método `POST`, y espera a que el documento PHP le devuelva los resultados de la consulta.
- `loadVars_datos.php` obtiene los valores de todos los campos del registro cuyo `id` es 4 y los escribe en variables siguiendo el formato `&variable=valor`.
- `detalle_usuario` recibe correctamente los datos escritos por `loadVars_datos.php` y se ejecutan las acciones de la función asociada a su evento `onLoad`.
- La función asociada al evento `onLoad` del objeto `detalle_usuario` es `mostrarDetalle`, que formatea el texto contenido en las variables de `detalle_usuario` para ser mostrado en el campo de texto `detalles`.

Como ha podido comprobar, los documentos que hemos utilizado realizan su labor correctamente. Sin embargo, ¿por qué utilizar dos documentos PHP (uno para el listado y otro para los detalles)? En el siguiente apartado, realizaremos el mismo ejercicio pero utilizando un solo documento PHP que nos servirá para realizar tanto el listado como la obtención de detalles, con lo que ganaremos en claridad y eficiencia a la hora de trabajar.

Utilización del objeto *LoadVars* y un solo documento PHP

Elimine los documentos de la carpeta `phpflash` de su servidor local y copie allí los documentos que se encuentran en la carpeta `material/cap6/loadvars/carga_conjunta` del CD. Observe cómo en esta ocasión sólo disponemos de un documento PHP (además de los de la carpeta *includes*).

Antes de comenzar, abra en su navegador el documento `carga_conjunta.html` tecleando `http://localhost/phpflash/carga_conjunta.html` y observe cómo se muestran los usuarios y sus detalles de igual forma a como hicimos utilizando dos documentos PHP en el ejercicio anterior.

La idea principal es la de que vamos a utilizar un documento PHP que va a recibir una variable de nombre `hacer`, enviada por Flash mediante el método `POST`. Si el contenido de `hacer` es "todos", entonces el documento PHP escribirá el listado de `nick` e `id` de los usuarios de la tabla. Si, por el contrario, el contenido de `hacer` es "datos", el documento PHP escribirá los detalles del usuario cuyo identificador único coincida con el valor de la variable `id` que Flash también debe enviarle en este caso.

Comencemos analizando el documento `carga_conjunta fla`. Ábralo y encontrará con que el entorno es el mismo que en el ejercicio anterior. Las únicas diferencias se encuentran en las acciones que se han declarado en el fotograma 1 de la capa *acciones*.



Nota: Puesto que este documento .fla es prácticamente idéntico al del ejercicio anterior, se comentarán sólo aquellas líneas que hayan sido modificadas con respecto a aquél.

```
// declaración de variables a utilizar
inicioX=45;
inicioY=125;
separacionVertical=30;

// función que mostrará los detalles de cada usuario en el campo de texto de la
derecha
mostrarDetalle=function(exito){
    if (exito){
        if (this.output=="ok"){
            detalles.htmlText="";
            detalles.htmlText+="<b>id: </b>"+this.id+"<br>";
            detalles.htmlText+="<b>nombre: </b>"+this.nombre +
            "+this.apellido+"<br>";
            detalles.htmlText+="<b>nick: </b>"+this.nick+"<br>";
            detalles.htmlText+="<b>email: </b><u><a
href=\"mailto:\"+this.email+\"\">"+this.email+"</a></u><br>";
            detalles.htmlText+="<b>url: </b><u><a href=\"\"+this.url+\"\"
target=\"_blank\">"+this.url+"</a></u><br>";
        } else {
            detalles.htmlText="Se produjo el siguiente error: <b>"+this.msg+"</
b>";
        }
    } else {
        detalles.htmlText="Error al cargar los datos del usuario";
    }
}
```

El código visto hasta el momento es idéntico al del documento .fla del ejercicio anterior. Sin embargo, hay una pequeña modificación en la función `obtenerDetalle`, y es el hecho de que antes de llamar al documento PHP, además de crear un atributo de nombre `id` para el objeto `detalle_usuario`, también se crea

un atributo de nombre `hacer`, en el que almacenamos la cadena de texto "datos", para que cuando el documento PHP reciba la llamada y obtenga ese dato enviado mediante POST, sepa que ha de realizar la consulta referente a obtener los detalles del usuario cuyo `id` es también enviado:

```
// función que extrae los detalles del usuario cuyo id se pasa por parámetro
obtenerDetalle=function(id){
    var detalle_usuario=new LoadVars();
    detalle_usuario.hacer="datos";
    detalle_usuario.id=id;
    detalle_usuario.onLoad=mostrarDetalle;
    detalle_usuario.sendAndLoad("http://localhost/phpflash/
loadVars_juntos.php",detalle_usuario,"POST");
}

// objeto LoadVars que recibe el listado de usuarios
datos_bd=new LoadVars();

// configuración de la variable que sirve para elegir la actividad del PHP:
// "todos" sirve para obtener listado de usuarios
// "datos" sirve para obtener detalles de un usuario
datos_bd.hacer="todos";
```

El objeto *LoadVars* que almacenará el listado de usuarios, *datos_bd*, también se crea de la misma forma que antes, pero además se le añade un atributo llamado *hacer* en el que se guarda la cadena de texto "todos", para que

cuando el documento PHP reciba la llamada y obtenga ese dato enviado mediante POST, sepa que ha de realizar la consulta referente a obtener todos los *nick* e *id* de los usuarios de la tabla.

```
// función que procesa la información del listado de usuarios
datos_bd.onLoad=function(exito){
    if(exito){
        if(this.output=="ok"){
            this.total=parseInt(this.total);
            for(var n=1;n<this.total+1;n++){
                var nom=_root.attachMovie("nombre","nombre"+n,n+1,
{_x:inicioX,_y:inicioY+(separacionVertical*(n-1))});
                nom.nick.text=this["nick"+n];
                nom.id=this["id"+n];
                nom.onRelease = function (){
                    this._parent.obtenerDetalle(this.id);
                }
            }
            detalles.htmlText="Elija un usuario en el menú de la izquierda.";
        }else{
            detalles.htmlText="Se produjo el siguiente error: <b>"+this.msg+"</b>";
        }
    }else{
        detalles.htmlText="<b>Error al cargar los datos</b>";
    }
}
```

El resto del código permanece intacto hasta que llegamos a la última línea, en la que en vez de utilizar el método `load` del objeto *LoadVars*, utilizamos el método `sendAndLoad`, ya que no sólo hemos de acceder a los contenidos que genera el documento PHP, sino que para que éste realice sus acciones, previamente hemos de enviar información (hacer) utilizando el método `POST`:

```
// acceso a la base de datos para obtener un listado de usuarios
datos_bd.sendAndLoad("http://localhost/phpflash/
loadVars_juntos.php",datos_bd,"POST");
```

Edite ahora el documento `loadVars_juntos.php`, en el que encontrará el siguiente código:

```
<?php
if(!empty($_POST['hacer'])) {
```

Primeramente, comprobamos que la variable `hacer` ha sido enviada mediante `POST` y contiene datos. Si es así, añadimos los documentos que almacenan las variables y funcio-

nes de uso común y establecemos una conexión con la base de datos:

```
include ("includes/config.php");
include ("includes/
funciones.php");
```

```
$cnx = conectar();
```

Ahora comprobamos el contenido de la variable `hacer`. Si contiene la cadena de texto `"todos"`, extraemos todos los datos tal y como hacía el documento `loadVars_ids.php` en el apartado anterior:

```
if($_POST['hacer'] == "todos"){
    $sql = "SELECT id,nick FROM directorio";
    $res = mysql_query($sql) or die("output=error&msg=".mysql_error());
    if(mysql_num_rows($res) > 0){
        $salida = "&output=ok&";
        $contador = 0;
        while(list($id,$nick)= mysql_fetch_array($res)){
            //aumentamos en 1 el contador
            $contador++;
        }
        $salida.="id$contador=$id&nick$contador=".utf8_encode($nick)."&";
    }
    $salida.="total=$contador&";
    echo $salida;
    mysql_free_result($res);
    mysql_close($cnx);
    exit;
} //fin del if de ids.
```

En cambio, si la variable `hacer` contiene la cadena de texto "datos", hemos de revisar que la variable `id` haya sido enviada mediante POST y que contenga información. Si es así, se realiza la consulta para extraer los detalles del usuario cuyo identificador único coincide con el valor almacenado en `id`, justo la tarea que desarrollaba en el ejercicio anterior el documento `loadVars_datos.php`:

```
if($_POST['hacer'] == "datos"){
    if(!empty($_POST['id'])){
        $sql = "SELECT * FROM directorio WHERE id=".$_POST['id'];
        $res = mysql_query($sql) or die("output=error&msg=".mysql_error());
        if(mysql_num_rows($res) > 0){
            $salida = "&output=ok&";
            while($fila = mysql_fetch_array($res)){
                $salida.="id=".$_fila['id'];
                $salida.="&nombre=".utf8_encode($fila['nombre']);
                $salida.="&apellido=".utf8_encode($fila['apellido']);
                $salida.="&nick=".utf8_encode($fila['nick']);
                $salida.="&email=".$_fila['email'];
                $salida.="&url=".$_fila['url']."&";
            }
            echo $salida;
            mysql_free_result($res);
            mysql_close($cnx);
        }else{
            echo "output=error&msg=No hay datos";
        }
    }else{
```

Se llega a este punto si existen problemas con la variable `id` (que no existe o bien no ha sido enviada mediante POST), con lo que el documento PHP escribe un mensaje de error al respecto:

```
        echo "output=error&msg=Variables requeridas no encontradas";
    }
} //fin del if de datos.
}else{
```

Se llega a este punto si existen problemas con la variable `hacer` (que no existe o bien no ha sido enviada mediante POST), con lo que el

documento PHP escribe un mensaje de error al respecto.

```

echo "output=error&msg=Variables requeridas no encontradas";
}
?>

```

El código completo del documento

loadVars_juntos.php es el siguiente:

```

<?php
if(!empty($_POST['hacer'])) {
    include ("includes/config.php");
    include ("includes/funciones.php");

    //nos conectamos a mysql.
    $cnx = conectar();
    //revisamos el contenido de variable hacer.
    //si hacer = todos, sacamos todos los datos.
    if($_POST['hacer'] == "todos") {
        //consulta sql.
        $sql = "SELECT id,nick FROM directorio";
        //ejecutamos la consulta sql
        $res = mysql_query($sql) or die("output=error&msg=".mysql_error());
        //contamos el número de filas en el resultado.
        if(mysql_num_rows($res) > 0) {
            //si hay datos.
            $salida = "&output=ok&";
            $contador = 0;
            //parseamos la información guardándola en $salida.
            while(list($id,$nick)= mysql_fetch_array($res)) {
                //aumentamos en 1 el contador
                $contador++;
            }
            $salida.="id$contador=$id&nick$contador=".utf8_encode($nick)."&";
        }
        //agregamos el total de registros a la salida.
        $salida.="total=$contador&";
        //imprimimos la salida.
        echo $salida;
        //liberamos memoria
        mysql_free_result($res);
        //cerramos la conexión
        mysql_close($cnx);
        //detenemos la ejecución de la página.
        exit;
    } //fin del if de ids.

    //si hacer = datos.
    if($_POST['hacer'] == "datos") {
        //revisamos que exista la variable id.
        if(!empty($_POST['id'])) {
            //si hay id, seguimos.
            // consulta sql.
            $sql = "SELECT * FROM directorio WHERE id=".$_POST['id'];
            //ejecutamos la consulta sql
            $res = mysql_query($sql) or die("output=error&msg=".mysql_error());

```

```
//contamos el número de filas en el resultado.
if(mysql_num_rows($res) > 0){
    //si hay datos.
    $salida = "&output=ok&";
    //parseamos la información guardándola en $salida.
    while($fila = mysql_fetch_array($res)){
        $salida.="id=".$fila['id'];
        $salida.="&nombre=".utf8_encode($fila['nombre']);
        $salida.="&apellido=".utf8_encode($fila['apellido']);
        $salida.="&nick=".utf8_encode($fila['nick']);
        $salida.="&email=".$fila['email'];
        $salida.="&url=".$fila['url']."&";
    }
    //imprimimos la salida.
    echo $salida;
    //liberamos memoria
    mysql_free_result($res);
    //cerramos la conexión
    mysql_close($cnx);
}else{
    //no hay datos, pasamos el mensaje a flash.
    echo "output=error&msg=No hay datos";
}
}else{
    //no hay id, error.
    echo "output=error&msg=Variables requeridas no encontradas";
}
}
}
//fin del if de datos.
```

La utilización de *loadVars* es perfectamente válida para acceder a los contenidos dinámicos que pueden obtenerse de una base de datos, pero cuando hemos de mostrar muchos resultados o bien información jerárquica, hay ocasiones en que utilizar el formato `&variable=valor` resulta un tanto desorganizado. En el siguiente apartado de este capí-

tulo, aprovecharemos las posibilidades de que dispone Flash a la hora de gestionar contenidos escritos en lenguaje de marcado XML para que nuestros documentos PHP utilicen este último para escribir los resultados de sus consultas.

Obtención de datos utilizando el objeto *XML*

En el capítulo 1 del libro analizamos cómo obtener información almacenada en documentos escritos en lenguaje de marcado XML. Si recuerda, cada uno de estos documentos se estructuraban mediante nodos en una estructura de árbol, como los del ejemplo que describimos:

```
<anuncios>
<coches>
<auto marca="Yoyota" anio="2001" />
<auto fecha="Yuntai" anio="2002" />
</coches>
<viviendas>
<piso calle="Margaritas" numero="25" />
<piso calle="Rosas Silvestres" numero="32" />
<piso calle="Plaza Amapolas" numero="s/n" />
</viviendas>
</anuncios>
```

Si decidimos no usar saltos de línea, el documento XML se organizaría de la siguiente forma:

```
<anuncios><coches><auto marca="Yoyota" anio="2001" /><auto fecha="Yuntai"
anio="2002" /></coches><viviendas><piso calle="Margaritas" numero="25" /><piso
calle="Rosas Silvestres" numero="32" /><piso calle="Plaza Amapolas" numero="s/
n" /></viviendas></anuncios>
```

Posteriormente, hacíamos uso del objeto predeterminado XML de Flash MX para acceder al árbol XML y extraer la información para su proceso.

De igual forma en que anteriormente utilizábamos los documentos PHP para escribir los datos en un formato que pudiera entender el objeto *LoadVars*, en este apartado aprenderemos a utilizarlos para que escriban los resultados de sus consultas utilizando lenguaje XML, con objeto de que la información pueda ser accedida e interpretada por el objeto XML de Flash MX.

Ejemplo básico

Para ver un ejemplo básico de funcionamiento, utilizaremos los documentos contenidos en la carpeta `material/cap6/xml/basico` del CD. Cópielos en la carpeta `phpflash` de su servidor local (tras haber eliminado previamente los documentos contenidos en ésta) y edite el documento `un_xml.php`, en el que encontrará este código:

```
<?php
$nombre="Fernando";
$url="http://www.onelx.com";
```

Definimos dos variables con un nombre (\$nombre) y una dirección Web (\$url), y ahora vamos a hacer que el documento PHP escriba esas dos variables utilizando lenguaje XML. Para ello, crearemos un nodo de nombre *datos* y estableceremos dos atributos para el mismo, nombre y url, a los que asignaremos el valor de las dos variables definidas al comienzo del código:

```
echo "<datos nombre='$nombre'
url='$url' />";
?>
```

Con lo que cuando se ejecute el documento PHP, la cadena de texto que se generará será la siguiente:

```
<datos nombre="Fernando" url="http://
/www.onelx.com" />
```

Una vez que hemos creado el documento PHP, abra el documento *basico fla* para observar cómo hemos accedido a los datos. En el fotograma 1 de la capa *acciones* encontrará el siguiente código:

```
datos_bd=new XML();
```

Creemos un objeto XML de nombre *datos_bd*, y posteriormente definimos las acciones para su evento *onLoad*, que se ejecutarán cuando el objeto realice la carga (con éxito o no) de los datos XML (escritos en este caso por el documento PHP):

```
datos_bd.onLoad=function(exito){
    if (exito){
```

Si la carga se realiza con éxito, simplemente mostramos los datos completos con objeto de mostrar que se accedieron correctamente. Si se produce algún error, mostramos un mensaje en la ventana de Salida:

```
        trace(this);
    } else {
        trace("Error al cargar los datos");
    }
}
```

Finalmente, el objeto XML realiza la llamada al documento PHP para que escriba los datos que necesitamos:

```
datos_bd.load("http://localhost/
phpflash/un_xml.php");
```

Si ahora reproduce su película, obtendrá la siguiente información en la ventana de Salida:

```
<datos nombre="Fernando"
url="http://www.onelx.com" />
```

Con lo que disponemos de un objeto conteniendo datos generados por un documento PHP y a los que podemos acceder mediante sus nodos y atributos. En el siguiente ejemplo utilizaremos un documento PHP para generar en lenguaje XML el listado de usuarios existentes en la tabla *directorio*.

Obtención del listado de usuarios

Elimine los documentos de la carpeta *phpflash* de su servidor local y copie allí los documentos que encontrará en la carpeta *material/cap6/xml/carga_simple* del CD. Utilizaremos un documento *.fla* con el mismo código que en el apartado anterior y un documento PHP que se conectará a la base de datos para obtener, escrito en lenguaje XML, un listado de todos los *nick* e *id* de la tabla

directorio. Este último será similar al que utilizábamos para obtener el listado en el formato que utiliza el objeto *LoadVars*, con la diferencia de que lo obtendremos en lenguaje XML. Por tanto, el texto que escribirá el documento PHP será:

```
<datos><registro id="1" nick="rolf"
/><registro id="2" nick="granatta" /
>...</datos>
```

en vez de:

```
&nick1=rolf&id1=1&nick2=granatta&id2=2...
```

Edite el documento `xml_ids.php`, en el que encontrará el siguiente código:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");
```

Tras incluir los documentos en los que se declaran las variables y funciones de uso común, establecemos la conexión con la base de datos MySQL y guardamos en la variable `$sql` la consulta que deseamos realizar, la selección de los contenidos de `nick` e `id` de todos los registros de la tabla *directorio*:

```
$cnx = conectar();
$sql = "SELECT id,nick FROM
directorio";
```

Ejecutamos la consulta SQL, almacenando el resultado en la variable `$res`, para comprobar en ésta el número de filas, con objeto de saber si la consulta ha retornado datos o no:

```
$res = mysql_query($sql) or
die("output=error&msg=".mysql_error());
if(mysql_num_rows($res) > 0){
```

Si existen datos, abrimos la etiqueta principal del árbol XML, que contendrá la información:

```
$salida = "<datos>\n";
```

Y recorremos el contenido de la variable `$res` concatenando la información junto con la obtenida previamente en la variable `$salida`. Observe cómo de nuevo escribimos en formato UTF-8 los contenidos del campo `nick`, susceptible de contener acentos y/o caracteres especiales:

```
while(list($id,$nick)=
mysql_fetch_array($res)){
    $salida.="<t<registro
id='$id' nick
='".utf8_encode($nick)."'>\n";
}
```

Cuando hemos terminado de recoger la información contenida en la variable `$res`, cerramos la etiqueta principal:

```
$salida.="</datos>";
```

Finalmente, imprimimos la salida, liberamos la memoria y cerramos la conexión con la base de datos:

```
echo $salida;
mysql_free_result($res);
mysql_close($cnx);

}else{
```

Si no se obtienen datos tras la consulta se avisa de tal circunstancia:

```
echo "output=error&msg=No hay
datos";
}
?>
```

El código completo del documento

xml_ids.php es el siguiente:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");

//nos conectamos a mysql.
$cnx = conectar();
// consulta sql.
$sql = "SELECT id,nick FROM directorio";
//ejecutamos la consulta sql
$res = mysql_query($sql) or die("output=error&msg=".mysql_error());
//contamos el número de filas en el resultado.
if(mysql_num_rows($res) > 0){
    //si hay datos.
    //abrimos la etiqueta principal que contendrá los datos.
    $salida = "<datos>\n";
    //parseamos la información guardándola en $salida.
    while(list($id,$nick)= mysql_fetch_array($res)){
        $salida.="<t<registro id='".$id' nick ='".utf8_encode($nick)."'>\n";
    }
    //cerramos la etiqueta principal
    $salida.="</datos>";
    //imprimimos la salida.
    echo $salida;
    //liberamos memoria
    mysql_free_result($res);
    //cerramos la conexión
    mysql_close($cnx);
}else{
    //no hay datos, pasamos el mensaje a flash.
    echo "output=error&msg=No hay datos";
}
?>
```

Abra el documento xml_simple fla y vaya al fotograma 1 de la capa *acciones*, en el que encontrará este código, exactamente igual al

del ejercicio anterior (excepto en el nombre del documento PHP al que realiza la llamada el objeto XML):

```
datos_bd=new XML();

datos_bd.onLoad=function(exito){
    if (exito){
        trace(this);
    } else {
        trace("Error al cargar los datos");
    }
}

datos_bd.load("http://localhost/phpflash/xml_ids.php");
```

Si ahora reproduce su película, obtendrá correctamente organizada una estructura escrita en lenguaje XML con el listado de usuarios en la ventana de Salida:

```
<datos>
  <registro id="1"  nick="rolf"  />
  <registro id="2"  nick="granatta" />
  <registro id="3"  nick="fernando" />
  <registro id="4"  nick="sangles" />
  <registro id="5"  nick="jesús" />
  <registro id="6"  nick="kali" />
  <registro id="7"  nick="kadazuro" />
  <registro id="8"  nick="dasso" />
  <registro id="10" nick="kali" />
</datos>
```

Puede establecer como true el atributo ignoreWhite del objeto XML datos_bd para no interpretar como nodos vacíos los saltos de línea y los espacios entre los distintos nodos:

```
datos_bd=new XML();
datos_bd.ignoreWhite=1;

datos_bd.onLoad=function(exito){
  if (exito){
    trace(this);
  } else {
    trace("Error al cargar los datos");
  }
}

datos_bd.load("http://localhost/phpflash/xml_ids.php");
```

Con lo que si vuelve a reproducir su película, obtendrá de nuevo el listado en su ventana de Salida, pero esta vez escrito en una sola línea:

```
<datos><registro id="1"  nick="rolf"  /><registro id="2"  nick="granatta" />
<registro id="3"  nick="fernando" /><registro id="4"  nick="sangles" /><registro
id="5"  nick="jesús" /><registro id="6"  nick="kali" /><registro id="7"
nick="kadazuro" /><registro id="8"  nick="dasso" /><registro id="10"  nick="kali"
/></datos>
```

Una vez que disponemos de un documento PHP que genera un listado de los usuarios, crearemos a continuación otro documento PHP que se encargue de proporcionarnos los detalles de cada uno de ellos.

Utilización del objeto XML y dos documentos PHP

Elimine los documentos de la carpeta phpflash de su servidor local y copie allí los que encontrará en la carpeta material/cap6/xml/carga_separada del CD. Abra en su navegador el documento xml_separados.html tecleando `http://localhost/phpflash/xml_separados.html`. Al

igual que al trabajar con el objeto *LoadVars*, el contenido que verá ha de ser un listado de *nicks* de los usuarios de la tabla *directorio* ordenados de forma ascendente por su identificador *id*. Para conseguir esta información, hemos utilizado el documento *xml_ids.php*, que es exactamente igual al del ejercicio anterior, por lo que su función también es la misma, escribir los contenidos de los campos *nick* e *id* de cada uno de los registros de la tabla en lenguaje XML. Flash accederá a esos contenidos y los mostrará en la forma que puede verse en la figura 6.6.

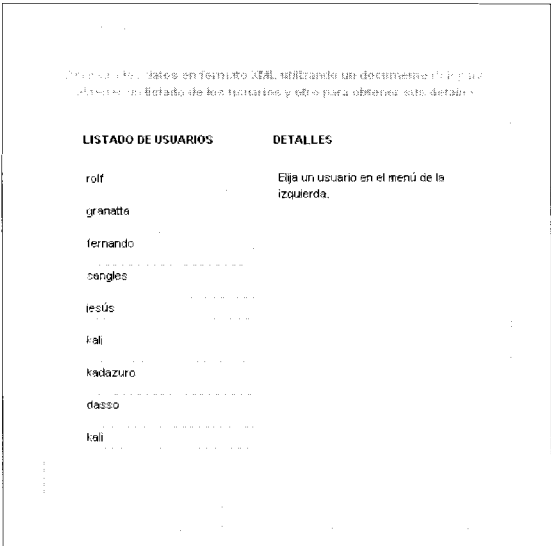


Figura 6.6.
Listado de usuarios tras la consulta a la base de datos.

Si ahora pulsamos sobre uno de los usuarios, por ejemplo, *jesús*, obtendremos información más detallada sobre el mismo en la columna derecha de la pantalla. Al pulsar sobre un usuario, Flash envía mediante el método GET su identificador *id* al llamar al otro documento PHP de que disponemos, *xml_datos.php*.

La función de este documento es la de realizar una consulta a la base de datos basándose en la variable *id* cuyo valor se recibe mediante el método GET, escribiendo entonces el contenido de los campos *nombre*, *apellido*, *email* y *url* del registro de la tabla cuyo identificador único coincide con el valor de la variable *id*. Flash accederá a esos contenidos y los mostrará como puede verse en la figura 6.7.

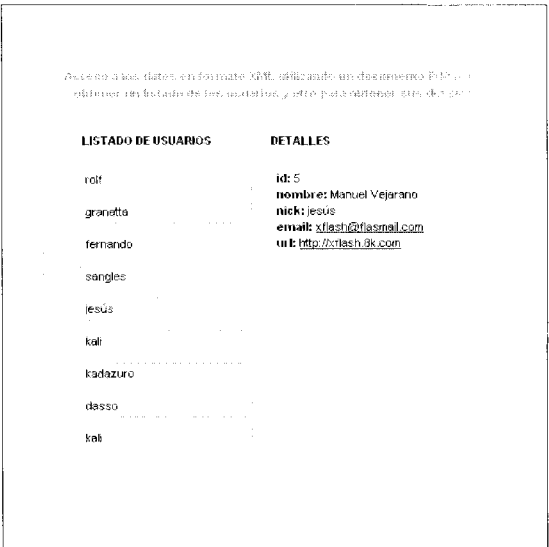


Figura 6.7.
Detalles del usuario jesús.

Edite este documento (*xml_datos.php*), en el que encontrará el siguiente código:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");
```

Incluimos los documentos que contienen las variables y funciones de uso común y nos conectamos a la base de datos MySQL:

```
$cnx = conectar();
```

Definimos la consulta que deseamos realizar y la almacenamos en la variable `$sql`, para posteriormente efectuar la consulta (en este caso seleccionar todos los campos del registro cuyo identificador único (`id`) coincida con el valor de la variable `id` enviada mediante GET) y contar el número de filas almacenadas en la variable que almacena el resultado (`$res`) para saber si la consulta retornó contenidos:

```
$sql = "SELECT * FROM directorio WHERE id=".$_GET['id'];
$res = mysql_query($sql) or
die("output=error".$_GET['id']."&msg=mysql_error()");
if(mysql_num_rows($res) > 0){
```

Si existen datos, abrimos la etiqueta principal del árbol XML, que contendrá la información:

```
$salida .= "<datos>\n";
```

Recorremos los contenidos almacenados en la variable `$res` y los concatenamos en la variable `$salida`, con lo que, al finalizar, cerramos la etiqueta principal e imprimimos el texto de salida:

```
while($fila = mysql_fetch_array($res)){
    $salida .= "\t<registro ";
    $salida.="id='".$_GET['id']."'";
    $salida.=" nombre='".utf8_encode($fila['nombre'])."'";
    $salida.=" apellido='".utf8_encode($fila['apellido'])."'";
    $salida.=" nick='".utf8_encode($fila['nick'])."'";
    $salida.=" email='".$fila['email']."'";
    $salida.=" url='".$fila['url']."'"/>\n";
}
$salida.="</datos>";
echo $salida;
```

Para finalizar, liberamos la memoria y cerramos la conexión con la base de datos:

```
mysql_free_result($res);
mysql_close($cnx);

}else{
```

Si no se obtienen datos tras la consulta, se avisa de tal circunstancia:

```
echo "output=error&msg=No hay datos";
}
?>
```

Cuando este documento PHP sea llamado, recibirá mediante el método GET un valor de `id`. Por ejemplo, al pulsar sobre el usuario *jesús* se envía el valor 5, con lo que el documento PHP escribirá el siguiente texto de salida:

```
<datos><registro id="5" nombre="Manuel" apellido="Vejarano" nick="jesús"
email="xflash@flasmail.com" url="http://xflash.8k.com" /></datos>
```

El código completo del documento

xml_datos.php es el siguiente:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");

//nos conectamos a mysql.
$cnx = conectar();
// consulta sql.
$sql = "SELECT * FROM directorio WHERE id=".$_GET['id'];
//ejecutamos la consulta sql
$res = mysql_query($sql) or
die("output=error".$_GET['id']."&msg=mysql_error()");
//contamos el número de filas en el resultado.
if(mysql_num_rows($res) > 0){
    //si hay datos.
    //abrimos la etiqueta principal que contendrá los datos.
    $salida .= "<datos>\n";
    //parseamos la información guardándola en $salida.
    while($fila = mysql_fetch_array($res)){
        $salida .= "\t<registro ";
        $salida.="id='". $fila['id']. "'";
        $salida.=" nombre='".utf8_encode($fila['nombre']). "'";
        $salida.=" apellido='".utf8_encode($fila['apellido']). "'";
        $salida.=" nick='".utf8_encode($fila['nick']). "'";
        $salida.=" email='". $fila['email']. "'";
        $salida.=" url='". $fila['url']. "' />\n";
    }
    //cerramos la etiqueta principal
    $salida.="</datos>";
    //imprimimos la salida.
    echo $salida;
    //liberamos memoria
    mysql_free_result($res);
    //cerramos la conexión
    mysql_close($cnx);
}else{
    //no hay datos, pasamos el mensaje a flash.
    echo "output=error&msg=No hay datos";
}
?>
```

Ahora veremos cómo Flash recoge estos datos para mostrarlos en pantalla. Abra el documento `xml_separados fla`, en cuyo primer fotograma encontrará una disposición de

contenidos idéntica a la de los ejercicios que realizamos utilizando el objeto *LoadVars*. Vaya a la capa *acciones*, en la que encontrará el siguiente código:

```
// declaración de variables a utilizar
inicioX=45;
inicioY=125;
separacionVertical=30;
```

Declaramos las mismas variables que en los ejercicios anteriores y, a continuación, establecemos como `true` el atributo `ignoreWhite` en la definición del objeto XML, con lo que a partir de este momento, todas las instancias que creamos de este objeto dispondrán del valor `true` en ese atributo:

```
XML.prototype.ignoreWhite=true;
```

La siguiente función será la que mostrará los detalles de cada usuario en el campo de texto de la derecha. Puesto que está asociada a una instancia del objeto XML, cuando utilizamos `this` estamos refiriéndonos a dicho objeto, cuya estructura será la que Flash recibe cuando se solicitan los detalles de un usuario realizando una llamada a `xml_datos.php`. En el caso del usuario *jesús*, la función habría de mostrar los datos referentes al texto:

```
<datos><registro id="5" nombre="Manuel" apellido="Vejarano" nick="jesús"
email="xflash@flasmail.com" url="http://xflash.8k.com" /></datos>

// función que mostrará los detalles de cada usuario en el campo de texto de la
derecha
mostrarDetalle=function(exito){
    if (exito){
```

Si el objeto XML accede con éxito a los datos, almacenaremos en la variable `datos_usuario` el primer hijo (`<registro>`) del nodo raíz (`<datos>`), y posterior-

mente accederemos a sus atributos (`id`, `nombre`, `nick`, `email` y `url`), formateándolos en lenguaje HTML para mostrarlos en el campo de texto detalles:

```
var datos_usuario=this.firstChild.childNodes[0];
detalles.htmlText="";
detalles.htmlText+="id: " +datos_usuario.attributes.id+ "<br>";
detalles.htmlText+="nombre: " +datos_usuario.attributes.nombre + "
"+datos_usuario.attributes.apellido+ "<br>";
detalles.htmlText+="nick: " +datos_usuario.attributes.nick+ "<br>";
detalles.htmlText+="email: " +datos_usuario.attributes.email+ "<br>";
detalles.htmlText+="url: " +datos_usuario.attributes.url+ "<br>";
detalles.htmlText+="target: " +datos_usuario.attributes.target+ "<br>";
} else {
    detalles.htmlText="Error al cargar los datos del usuario";
}
```

La siguiente función es llamada cada vez que se pulsa sobre un usuario. En ella se crea un objeto XML al que se asocia la función `mostrarDetalle` para su evento `onLoad`.

```
// función que extrae los detalles del usuario cuyo id se pasa por parámetro
obtenerDetalle=function(id){
    var detalle_usuario=new XML();
    detalle_usuario.onLoad=mostrarDetalle;
```

Finalmente, utilizamos el método `sendAndLoad` del objeto XML, cuya sintaxis es:

```
objetoXML.sendAndLoad(ruta_archivo_llamado, objetoXML_receptor);
```

donde `ruta_archivo_llamado` es la ruta del archivo PHP al que se llama y envían variables mediante el método `GET`, y `objetoXML_receptor` es el objeto XML que recibirá los resultados impresos por el documento PHP:

```
detalle_usuario.sendAndLoad("http://localhost/phpflash/
xml_datos.php?id="+id,detalle_usuario);
```

Finalmente, llamamos al documento `xml_datos.php` enviando el contenido de la variable `id` mediante el método `GET` y especificamos que los resultados sean retornados al mismo objeto XML que realizó la llamada.

```
}

// objeto XML que recibe el listado de usuarios
datos_bd=new XML();
```

Creamos un nuevo objeto XML que será el que reciba el listado de `nick` e `id` de los usuarios de la tabla. Posteriormente, definimos las acciones de su evento `onLoad`, que se ejecuta-

rán cuando el objeto realice la carga (exitosa o no) de los datos generados por el documento PHP:

```
// función que procesa la información del listado de usuarios
datos_bd.onLoad=function(exito){
    if(exito){
```

Si se realiza con éxito la carga de los datos impresos por `xml_ids.php`, almacenamos en la variable `raiz` el nodo raíz del árbol XML (`<datos>`), y su `array` de nodos hijos (cada uno de los nodos `<registro>`) en la variable `hijos`:

```
var raiz=this.firstChild;
var hijos=this.firstChild.childNodes.length;
```

Recorremos dicho `array`, importando por cada uno de sus elementos una instancia del clip nombre de la Biblioteca, colocándola en el escenario, asignando un valor (atributo `id`) para su variable `id` y rellenando su campo de texto con el contenido del atributo `nick`:

```
for(var n=0;n<hijos;n++){
    var
nom=_root.attachMovie("nombre","nombre"+n,n+1,{_x:inicioX,_y:inicioY+(separacionVertical*n)});
    nom.nick.text=raiz.childNodes[n].attributes.nick;
    nom.id=raiz.childNodes[n].attributes.id;
    nom.onRelease = function (){
        this._parent.obtenerDetalle(this.id);
    }
}
```

También añadimos un texto inicial para el campo de texto `detalles`:

```
detalles.htmlText="Elija un usuario en el menú de la izquierda.";
}else{
```

En caso de que no se puedan mostrar los datos porque se haya producido un error al acceder a los mismos, se muestra un mensaje avisando de tal circunstancia:

```
    detalles.htmlText="<b>Error al cargar los datos</b>";
}

// acceso a la base de datos para obtener un listado de usuarios
datos_bd.load("http://localhost/phpflash/xml_ids.php");
```

Al igual que en el ejercicio anterior, llamamos al documento `xml_ids.php` para obtener el listado de usuarios.

En este ejercicio hemos utilizado un documento PHP para obtener el listado de usuarios y otro documento PHP para obtener los detalles de cada uno de ellos. En el próximo ejercicio utilizaremos únicamente un documento PHP, cuyo cometido será el de escribir el árbol XML completo referente a los registros de la tabla *directorio*.

Utilización del objeto XML y un documento PHP

Elimine los documentos de la carpeta `phpactual` de su servidor local y copie allí los documentos que se encuentran en la carpeta `material/cap6/xml/carga_total` del CD. Abra en su navegador el documento `xml_total.html` tecleando `http://localhost/phpflash/xml_total.html` y observe cómo la funcionalidad del ejercicio es la misma que en el apartado anterior.

Primeramente, veremos cómo se genera el texto XML con los datos de la tabla. Edite el documento `xml_todo.php`, en el que encontrará el siguiente código:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");

//nos conectamos a mysql.
$cnx = conectar();
// consulta sql.
$sql = "SELECT * FROM directorio";
//ejecutamos la consulta sql
$res = mysql_query($sql) or die("output=error&msg=".mysql_error());
//contamos el número de filas en el resultado.
if(mysql_num_rows($res) > 0){
Hasta este punto seguimos el mismo proceso que en los ejercicios anteriores.
Una vez que encontramos datos en la variable $res, concatenamos los contenidos
de los distintos campos de los registros en la variable $salida:
    $salida = "<datos>\n";
    while($fila = mysql_fetch_array($res)){
        $salida .= "\t<registro ";
        $salida.="id='".$fila['id']."'";
        $salida.=" nombre='".$utf8_encode($fila['nombre'])."'";
        $salida.=" apellido='".$utf8_encode($fila['apellido'])."'";
        $salida.=" nick='".$utf8_encode($fila['nick'])."'";
        $salida.=" email='".$fila['email']."'";
        $salida.=" url='".$fila['url']."'>\n";
    }
}
```

Cerramos la etiqueta principal e imprimimos la salida, finalizando con la liberación de memoria y el cierre de conexión:

```
$salida.="</datos>";
echo $salida;
mysql_free_result($res);
mysql_close($cnx);

}else{
```

Si no se obtienen datos tras la consulta, se avisa de tal circunstancia:

```
echo "output=error&msg=No hay
datos";
}
?>
```

Cuando este documento .php es llamado, éste es el texto que genera (de acuerdo a los datos que actualmente están contenidos en la tabla *directorio*):

```
<datos>
<registro id="1" nombre="Rodolfo" apellido="Ruiz" nick="rolf"
email="rolf@alesys.net" url="http://www.alesys.net" />
<registro id="2" nombre="Daniel" apellido="de la Cruz" nick="granatta"
email="null@granatta.com" url="http://www.granatta.com" />
<registro id="3" nombre="Fernando" apellido="Flórez" nick="fernando"
email="info@onelx.com" url="http://www.onelx.com" />
<registro id="4" nombre="Santiago" apellido="Anglés" nick="sangles"
email="info@sangles.com" url="http://www.sangles.com" />
<registro id="5" nombre="Manuel" apellido="Vejarano" nick="jesús"
email="xflash@flasmal.com" url="http://xflash.8k.com" />
<registro id="6" nombre="Kali" apellido="Romiglia" nick="kali"
email="kali@romiglia.com" url="http://www.romiglia.com" />
<registro id="7" nombre="Carlos" apellido="Zumbado" nick="kadazuro"
email="nadie@kadazuro.com" url="http://www.kadazuro.com" />
<registro id="8" nombre="Ricardo" apellido="Salazar" nick="dasso"
email="info@nomaster.com" url="http://www.nomaster.com" />
<registro id="10" nombre="Kali" apellido="Romiglia" nick="kali"
email="kali@romiglia.com" url="http://www.romiglia.com" />
</datos>
```

El código completo del documento

xml_todo.php es el siguiente:

```
<?php
include ("includes/config.php");
include ("includes/funciones.php");

//nos conectamos a mysql.
$cnx = conectar();
// consulta sql.
$sql = "SELECT * FROM directorio";
//ejecutamos la consulta sql
$res = mysql_query($sql) or die("output=error&msg=".mysql_error());
//contamos el número de filas en el resultado.
if(mysql_num_rows($res) > 0){
    //si hay datos.
    //abrimos la etiqueta principal que contendrá los datos.
    $salida = "<datos>\n";
    //parseamos la información guardándola en $salida.
    while($fila = mysql_fetch_array($res)){
        $salida .= "\t<registro ";
        $salida.="id='".$fila['id']."'";
        $salida.=" nombre='".$utf8_encode($fila['nombre'])."'";
        $salida.=" apellido='".$utf8_encode($fila['apellido'])."'";
        $salida.=" nick='".$utf8_encode($fila['nick'])."'";
        $salida.=" email='".$fila['email']."'";
        $salida.=" url='".$fila['url']."'>\n";
    }
    //cerramos la etiqueta principal
    $salida.="</datos>";
    //imprimimos la salida.
    echo $salida;
    //liberamos memoria
    mysql_free_result($res);
    //cerramos la conexión
    mysql_close($cnx);
}else{
    //no hay datos, pasamos el mensaje a flash.
    echo "output=error&msg=No hay datos";
}
?>
```

Abra ahora el documento xml_total fla.
El entorno es idéntico a los anteriores ejerci-

cios, pero encontraremos diferencias en el
fotograma 1 de la capa *acciones*, que contiene el
siguiente código:

```
// declaración de variables a utilizar
inicioX=45;
inicioY=125;
separacionVertical=30;

XML.prototype.ignoreWhite=true;
```

La principal diferencia con respecto a la anterior versión que analizamos en el ejercicio anterior es la de que, puesto que el documento PHP imprime todos los campos de cada usuario, no necesitamos utilizar la función `obtenerDetalle`, sino que cuando pulsemos

sobre un usuario mostraremos directamente sus detalles mediante la función `mostrarDetalle`, que recibe como parámetro un valor de `id` que se utilizará para acceder a los atributos de uno de los nodos hijos del nodo raíz del árbol XML:

```
// función que mostrará los detalles de cada usuario en el campo de texto de la
derecha
mostrarDetalle=function(id){
    var datos_usuario=datos_bd.firstChild.childNodes[id];
    detalles.htmlText="";
    detalles.htmlText+="id:  </b>" +datos_usuario.attributes.id+"<br>";
    detalles.htmlText+="nombre:  </b>" +datos_usuario.attributes.nombre  +"
"+datos_usuario.attributes.apellido+"<br>";
    detalles.htmlText+="nick:  </b>" +datos_usuario.attributes.nick+"<br>";
    detalles.htmlText+="email:  </b><u><a
href=\"mailto:\"+datos_usuario.attributes.email+\">\"+datos_usuario.attributes.email+\"</
a></u><br>";
    detalles.htmlText+="url:  </b><u><a
href=\"\"+datos_usuario.attributes.url+\"\"
target=\"_blank\">\"+datos_usuario.attributes.url+\"</a></u><br>";
}

// objeto XML que recibe el listado de usuarios
datos_bd=new XML();

// función que procesa la información del listado de usuarios
datos_bd.onLoad=function(exito){
    if(exito){
        var raiz=this.firstChild;
        var hijos=this.firstChild.childNodes.length;
        for(var n=0;n<hijos;n++){
            var
nom=_root.attachMovie("nombre","nombre"+n,n+1,{_x:inicioX,_y:inicioY+(separacionVertical*n)});
            nom.nick.text=raiz.childNodes[n].attributes.nick;
            nom.id=n;
            nom.onRelease = function (){
                this._parent.mostrarDetalle(this.id);
            }
        }
    }
}
```

De esta forma, si se pulsa sobre, por ejemplo, el usuario *dasso*, cuyo valor de la variable *id* al importar una instancia del clip de película *nombre* es 7 (el valor de *n* en el bucle *for* cuando se accede a sus datos), se realiza una llamada a la función *mostrarDetalle* con el valor 7 como parámetro. Dicha función buscará y mostrará en el campo de texto *detalles* los atributos del nodo `nodo_raiz.childNodes[7]`, los correspondientes al usuario *dasso*.



Nota: No hay que confundir el valor de *id* de cada usuario en la tabla *directorio* con el que se le asigna en Flash para realizar la llamada a la función *mostrarDetalle*.

```
}
detalles.htmlText="Elija un usuario en el menú de la izquierda.";
}else{
    detalles.htmlText="<b>Error al cargar los datos</b>";
}
}

// acceso a la base de datos para obtener un listado de usuarios
datos_bd.load("http://localhost/phpflash/xml_todo.php?rn=" + new
Date().getTime());
```

Finalmente se realiza la llamada al documento `xml_todo.php`, añadiéndole una variable que hemos creado de nombre *rn*, cuyo cometido es el de asegurarse de que cada una de las llamadas al documento PHP es única, evitando así que se carguen los datos que puedan encontrarse en la caché del navegador si ha accedido con anterioridad a los mismos. Por ello se acompaña la llamada al documento de un número que se genera mediante el método `getTime` del objeto predeterminado *Date*, que retorna el número de milisegundos transcurridos desde las 0 horas del 1 de Enero de 1970 (*Hora Universal*). De esta forma, nos aseguramos de que el número que acompañe a la llamada siempre será distinto, puesto que dependerá de la hora en que se realice.

El método de acceso a los datos que hemos utilizado en este ejercicio es perfectamente válido, pero presenta el inconveniente de que si existe una cantidad elevada de datos, puede ser problemático generar (por su tamaño) un texto escrito en XML tan frecuentemente. En el próximo apartado retomaremos los documentos que utilizábamos en el capítulo anterior para modificar los contenidos de la base de datos, con la particularidad de que, además de realizar su cometido, también se encargarán de generar físicamente un documento XML en el directorio en el que trabajamos. Este documento .xml será el que utilizará Flash para acceder a la información que posteriormente muestre en pantalla en vez de realizar una llamada a un documento PHP.

Obtención de datos desde un documento XML

Generar ficheros XML es muy útil cuando la información almacenada no cambia con mucha frecuencia, además del ahorro en peticiones a la base de datos cuando se requiera una determinada información.

Tomando como base el documento `xml_todo.php` que creamos en el ejercicio anterior, crearemos un nuevo documento PHP, pero en esta ocasión almacenaremos los

datos resultantes de las consultas en un documento XML en vez de mostrarlas en pantalla. Elimine los documentos de la carpeta `phpflash` de su servidor local y copie allí los documentos que encuentre en la carpeta `material/cap6/xml/carga_fichero` del CD.

Este nuevo documento PHP constará de dos secciones; primero implementaremos un formulario en lenguaje HTML en el que pediremos confirmación para generar el documento XML, utilizando el botón **submit** de dicho formulario como variable que utilizaremos posteriormente:

```
<form name="form1" method="post" action="<?echo $_SERVER['PHP_SELF'];?>">
  <table width="400" border="0" cellpadding="0" cellspacing="0">
    <tr>
      <td align="center">¿Actualizar fichero xml??</td>
    </tr>
    <tr>
      <td align="center"><input type="submit" name="submit" value="Actualizar."></td>
    </tr>
  </table>
</form>
```

La acción asignada al formulario es la de que se envíe al mismo documento en que se declara, utilizando para ello la variable de servidor `PHP_SELF`. En el interior del formulario hemos creado una tabla con dos filas; la primera de ellas contiene la pregunta de si deseamos actualizar el documento XML, y la segunda contiene el botón de envío (de nombre **submit**) con el texto "Actualizar" como etiqueta.

En el fragmento de código PHP, primeramente comprobamos la existencia de la variable `submit` (que debe ser enviada por el formulario mediante el método POST):

```
if(isset($_POST['submit'])) {
```

De ser así, incluimos los documentos que contienen las variables y funciones de uso común, establecemos la conexión con la base de datos y definimos la consulta SQL, almacenándola en la variable `$sql`:

```
include ("includes/config.php");
include ("includes/funciones.php");
//nos conectamos a mysql.
$cnx = conectar();
// consulta sql.
$sql = "SELECT * FROM directorio";
//ejecutamos la consulta sql
$res = mysql_query($sql) or die("output=error&msg=".mysql_error());
```

Igual que hicimos en el ejercicio anterior, la consulta SQL incluye todas las columnas de la tabla *directorio*, puesto que deseamos obtener toda la información de la misma ("SELECT * FROM directorio").

Tras almacenar el resultado de la consulta en la variable `$res`, comprobamos si existen datos concatenando la información en una variable de salida (`$salida`) mientras recorremos los resultados (en caso de que los haya) utilizando un bucle `while`:

```
if(mysql_num_rows($res) > 0){
    //si hay datos.
    //abrimos la etiqueta principal que contendrá los datos.
    $salida = "<datos>\n";
    //parseamos la información guardándola en $salida.
    while($fila = mysql_fetch_array($res)){
        $salida .= "\t<registro ";
        $salida.="id='".$fila['id']."'";
        $salida.=" nombre='".$utf8_encode($fila['nombre'])."'";
        $salida.=" apellido='".$utf8_encode($fila['apellido'])."'";
        $salida.=" nick='".$utf8_encode($fila['nick'])."'";
        $salida.=" email='".$fila['email']."'";
        $salida.=" url='".$fila['url']."'>\n";
    }
}
```

Posteriormente, cerramos la etiqueta principal del documento XML que se va a generar y que hemos abierto al inicializar la variable `$salida` (`$salida = "<datos>\n";`).

```
//cerramos la etiqueta principal
$salida.="</datos>";
```

Cuando disponemos de toda la información de la tabla almacenada en `$salida`, procedemos a crear el documento XML que contendrá los datos. Comenzamos asignando un nombre a este nuevo documento:

```
$nombreFichero = "directorio.xml";
```

Intentamos abrir el fichero guardando el apuntador a éste en la variable `$fp`. Si este documento no existe, se creará uno nuevo:

```
$fp = fopen($nombreFichero, "w");
```



Nota: Lo que deseamos realizar con la información de la tabla es actualizar el contenido del fichero XML; es decir, cada vez que el documento PHP cuyo código estamos analizando sea llamado, el contenido del documento XML será reemplazado en su totalidad por la nueva información. Éste es el motivo por el que se abre utilizando "w" como parámetro, situando el apuntador al inicio del documento.

Como la variable `$salida` almacena la información de la tabla ya escrita en formato XML, escribimos el contenido de la variable en el fichero y lo cerramos:

```
fwrite($fp, $salida);  
fclose($fp);
```

Por último, liberamos memoria y cerramos la conexión con la base de datos MySQL:

```
mysql_free_result($res);  
mysql_close($cnx);
```

Una vez que se han realizado todos los procesos que tienen que ver con la base de datos y la escritura de sus contenidos en el fichero XML, mostramos en pantalla un mensaje notificando la actualización del fichero:

```
echo "fichero $nombreFichero,  
actualizado";
```

Y para evitar que el formulario se muestre de nuevo una vez que el documento PHP ha terminado de realizar sus acciones, utilizamos la instrucción de salida:

```
exit;
```

Si no hay resultados en la consulta, mostramos un mensaje alternativo que informe de tal circunstancia:

```
}else{  
    echo "output=error&msg=No hay datos";  
}
```

Finalmente, cerramos la estructura condicional `if` que verifica la existencia de la variable `submit` enviada mediante el método POST:

```
}
```

El código completo (que puede encontrar en el documentomaterial/cap6/xml/carga_fichero/generar_xml.php) es el siguiente:

```
<?php
if(isset($_POST['submit'])) {
    include ("includes/config.php");
    include ("includes/funciones.php");

    //nos conectamos a mysql.
    $cnx = conectar();
    // consulta sql.
    $sql = "SELECT * FROM directorio";
    //ejecutamos la consulta sql
    $res = mysql_query($sql) or die("output=error&msg=".mysql_error());
    //contamos el número de filas en el resultado.
    if(mysql_num_rows($res) > 0){
        //si hay datos.
        //abrimos la etiqueta principal que contendrá los datos.
        $salida = "<datos>\n";
        //parseamos la información guardándola en $salida.=
        while($fila = mysql_fetch_array($res)){
            $salida .= "\t<registro ";
            $salida.="id='". $fila['id'] ."'";
            $salida.=" nombre='".utf8_encode($fila['nombre'])."'";
            $salida.=" apellido='".utf8_encode($fila['apellido'])."'";
            $salida.=" nick='".utf8_encode($fila['nick'])."'";
            $salida.=" email='". $fila['email'] ."'";
            $salida.=" url='". $fila['url'] ."'>\n";
        }
        //cerramos la etiqueta principal
        $salida.="</datos>";
        //creación del fichero.
        //nombre
        $nombreFichero = "directorio.xml";
        //lo abrimos
        $fp = fopen($nombreFichero,"w");
        //escribimos el contenido de $salida en el.
        fwrite($fp,$salida);
        //cerramos el archivo
        fclose($fp);
        //liberamos memoria
        mysql_free_result($res);
        //cerramos la conexión
        mysql_close($cnx);
        echo "archivo $nombreFichero, actualizado";
        exit;
    } else {
        //no hay datos, pasamos el mensaje a flash.
        echo "output=error&msg=No hay datos";
    }
}
?>

<form name="form1" method="post" action="<?echo $_SERVER['PHP_SELF'];?>">
<table width="400" border="0" cellpadding="0" cellspacing="0">
```

```

<tr>
  <td align="center">¿Actualizar fichero xml??</td>
</tr>
<tr>
  <td align="center"><input type="submit" name="submit" value="Actualizar."></td>
</tr>
</table>
</form>

```

Abra ahora en su navegador el documento `generar_xml.php` tecleando `http://localhost/phpflash/generar_xml.php`, en el que se le pedirá confirmación para actualizar el fichero `directorio.xml` (figura 6.8).

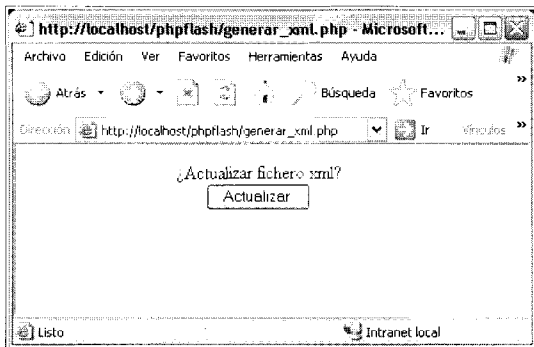


Figura 6.8.

Petición de confirmación para actualizar el documento `directorio.xml`.

Cuando presione el botón **Actualizar**, los contenidos del documento serán actualizados, lo que se notificará mediante un mensaje en su pantalla (figura 6.9).

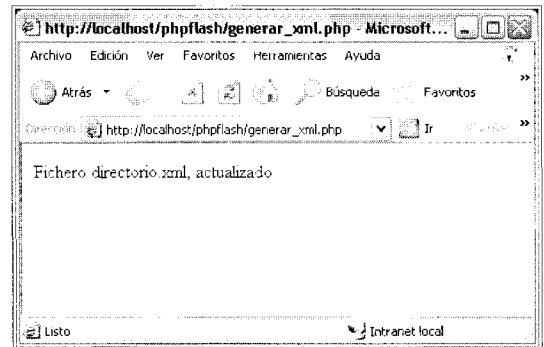


Figura 6.9.

Pantalla de notificación de actualización del documento `directorio.xml`.

Si ahora abre el documento `directorio.xml` tecleando en su navegador `http://localhost/phpflash/directorio.xml`, podrá observar el árbol XML de los contenidos de la tabla *directorio* (figura 6.10).

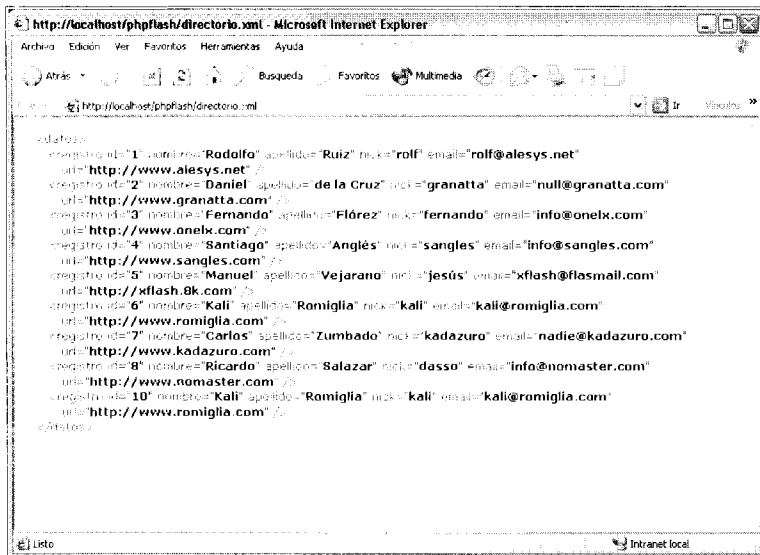


Figura 6.10.
*Contenidos del documento
directorio.xml.*

Para tener siempre actualizado el fichero `directorio.xml` con los contenidos de la tabla *directorio*, habremos de realizar una llamada al documento `generar_xml.php` cada vez que realicemos alguna modificación en los contenidos de la base de datos, o podemos actualizarlo automáticamente tras utilizar los documentos que se crearon en el capítulo 5 (`nuevo.php`, `editar.php`, `eliminar.php`).

Escribir el mismo código en cada uno de los cuatro documentos (los tres que hemos citado antes junto con el propio `generar_xml.php`) sería un tanto tedioso además de innecesario, puesto que vamos a crear una función que actualice el fichero `directorio.xml`, y posteriormente la incluiremos en el documento `funciones.php` de nuestra carpeta inclu-

des, para poder llamarla directamente cada vez que la base de datos sufra alguna modificación en sus contenidos.

Utilizaremos como referencia el código del documento `generar_xml.php` y lo transformaremos en una función. Necesitaremos también los documentos del capítulo anterior: `listado.php`, `ver.php`, `editar.php`, `nuevo.php` y `eliminar.php` (teniendo en cuenta que ya disponemos de los documentos `config.php` y `funciones.php` en la carpeta `phpflash/includes` del servidor local).

El código de la función, que será añadido a los contenidos del documento `funciones.php`, es el siguiente:

1/**

```

2función generar_xml
3genera el archivo directorio.xml
4**/
5function generar_xml() {
6    //nombre del xml.
7    $nombreFichero =
"directorio.xml";

```

Especificamos el nombre del fichero XML que vamos a generar. La razón de utilizar una variable para el nombre del fichero (\$nombreVariable) es la de que si incluimos el nombre del fichero en todos los documentos que invocarán a la función, cuando modifiquemos el nombre habremos de recorrer todos los documentos para realizar el cambio, mientras que de esta forma nos basta con modificar la variable \$nombreVariable del documento funciones.php.

```

8    //sql para el xml ( todo los datos )
9    $sqlFichero = "SELECT * FROM directorio";
10   //resultado de consulta de fichero.
11   $resFichero = mysql_query($sqlFichero) or die("No se puedo actualizar
fichero xml");

```

Creamos la consulta SQL y la ejecutamos, revisando posteriormente si hay datos en la variable \$resFichero:

```

12   //revisamos si hay datos.
13   if(mysql_num_rows($resFichero) > 0){
14       $salida = "<datos>\n";
15       //parseamos la información guardándola en $salida.
16       while($fila = mysql_fetch_array($resFichero)){
17           $salida .= "\t<registro ";
18           $salida.="id='".$fila['id']."'";
19           $salida.=" nombre='".$utf8_encode($fila['nombre'])."'";
20           $salida.=" apellido='".$utf8_encode($fila['apellido'])."'";
21           $salida.=" nick='".$utf8_encode($fila['nick'])."'";
22           $salida.=" email='".$fila['email']."'";
23           $salida.=" url='".$fila['url']."'>\n";
24       }
25       //cerramos la etiqueta principal
26       $salida.="</datos>";
27       //creación del fichero.
28       //lo abrimos
29       $fp = fopen($nombreFichero,"w");
30       //escribimos el contenido de $salida en el.
31       fwrite($fp,$salida);
32       //cerramos el archivo
33       fclose($fp);

```

Esta función será invocada cuando se inserten, modifiquen o eliminen registros de la tabla. En este último caso, hay que tener en cuenta que si no hay registros (si el número de filas en \$resFichero es 0), hay que eliminar el documento XML:

```
34 }else{
35     //si no hay datos borramos el xml.
36     unlink($nombreFichero);
37 }
38}
```

El código completo de la función es el siguiente:

```
1/**
2función generar_xml
3genera el archivo directorio.xml
4**/
5function generar_xml() {
6    //nombre del xml.
7    $nombreFichero = "directorio.xml";
8    //sql para el xml ( todo los datos )
9    $sqlFichero = "SELECT * FROM directorio";
10    //resultado de consulta de fichero.
11    $resFichero = mysql_query($sqlFichero) or die("No se puedo actualizar
fichero xml");
12    //revisamos si hay datos.
13    if(mysql_num_rows($resFichero) > 0){
14        $salida = "<datos>\n";
15        //parseamos la información guardándola en $salida.
16        while($fila = mysql_fetch_array($resFichero)){
17            $salida .= "\t<registro ";
18            $salida.="id='". $fila['id']. "'";
19            $salida.=" nombre='".utf8_encode($fila['nombre']). "'";
20            $salida.=" apellido='".utf8_encode($fila['apellido']). "'";
21            $salida.=" nick='".utf8_encode($fila['nick']). "'";
22            $salida.=" email='". $fila['email']. "'";
23            $salida.=" url='". $fila['url']. "'>\n";
24        }
25        //cerramos la etiqueta principal
26        $salida.="</datos>";
27        //creación del fichero.
28        //lo abrimos
29        $fp = fopen($nombreFichero,"w");
30        //escribimos el contenido de $salida en el.
31        fwrite($fp,$salida);
32        //cerramos el archivo
33        fclose($fp);
34    }else{
35        //si no hay datos borramos el xml.
36        unlink($nombreFichero);
37    }
38}
```

Ahora veremos la modificación en los documentos que actualizan el fichero `directorio.xml` haciendo uso de la función `generar_xml` que acabamos de crear. Hemos abierto el documento `generar_xml.php` y lo hemos guardado con el nombre de `generar_xml_funcion.php`. Dentro del mismo, hemos reemplazado todo el bloque de código PHP por este otro que realiza la llamada a la función `generar_xml`:

```
<?php
if(isset($_POST['submit'])){
    include ("includes/config.php");
    include ("includes/funciones.php");

    //nos conectamos a mysql.
    $cnx = conectar();
    generar_xml();
    echo "Fichero XML actualizado";
    mysql_close($cnx);
    exit;
}
?>
```

Por lo que el contenido del documento `generar_xml_funcion.php` es el siguiente:

```
1<?php
2if(isset($_POST['submit'])){
3    include ("includes/config.php");
4    include ("includes/funciones.php");
5
6    //nos conectamos a mysql.
7    $cnx = conectar();
8    generar_xml();
9    echo "Fichero XML actualizado";
10    mysql_close($cnx);
11    exit;
12}
13?>
14<form name="form1" method="post" action="<?echo $_SERVER['PHP_SELF'];?>">
15<table width="400" border="0" cellpadding="0" cellspacing="0">
16<tr>
17    <td align="center">¿Actualizar fichero xml??</td>
18</tr>
19<tr>
20    <td align="center"><input type="submit" name="submit" value="Actualizar."></td>
21</tr>
22</table>
23</form>
```

Una vez que se envía el formulario y se comprueba la existencia de la variable `submit`, establecemos una conexión con MySQL y llamamos a la función `generar_xml`, para posteriormente mostrar el mensaje de que el fichero se ha actualizado, cerrando después la conexión y deteniendo ejecuciones posteriores mediante el uso de `exit`.

Puesto que vamos a actualizar el documento XML haciendo uso de los documentos PHP de modificación (listado.php, ver.php, etc.) los documentos de generación del XML (generar_xml.php y generar_xml_funcion.php) no van a ser requeridos excepto en el caso de que realicemos modificaciones directamente sobre la base de datos (utilizando MySQL o phpMyAdmin).

Edita el documento nuevo.php de la carpeta phpflash de su servidor local. Tras la línea:

```
$res = mysql_query($sql) or  
die(mysql_error());
```

```
16 $sql = "INSERT INTO directorio ($campos) VALUES($valores)";  
17 $res = mysql_query($sql) or die(mysql_error());  
18 //actualizamos el xml.  
19 generar_xml();  
20 echo "Registro ingresado, y xml actualizado.<br><a  
href='listado.php'>regresar</a>";  
21 mysql_close($cnx);  
22 exit;
```

Hemos añadido la llamada a la función generar_xml:

```
generar_xml();
```

Al mostrar el mensaje de salida ("Registro ingresado ..."), agregaremos la información que indica que se ha actualizado el fichero XML.

```
echo "Registro ingresado, y xml  
actualizado.<br><a  
href='listado.php'>regresar</a>";
```

Desde la línea en que se produce la inserción de los nuevos datos hasta que se muestra el mensaje de salida, el código es el siguiente:

En el documento editar.php del mismo directorio, el proceso es similar; tras la línea en la que se ejecuta la consulta SQL se agrega la llamada a la función y la modificación al mensaje de salida.

```
17 $res = mysql_query($sql) or die(mysql_error());  
18 //actualizamos el xml.  
19 generar_xml();  
20 echo "Registro y xml actualizado.<br><a href='listado.php'>regresar</a>";  
21 mysql_close($cnx);  
22 exit;
```

También añadimos la llamada a la función y la modificación del mensaje en el documento `eliminar.php` del mismo directorio:

```
9  $sql = "DELETE FROM directorio WHERE id = ".$_POST['id'];
10 $res = mysql_query($sql) or die(mysql_error());
11 //actualizamos el xml.
12 generar_xml();
13 echo"Registro ".$_POST['id']." eliminado, XML actualizado<br><a
href='listado.php'>regresar</a>";
14 mysql_close($cnx);
15 exit;
```

Una vez que se han modificado los ficheros, abra el documento `listado.php` en su navegador (tecleando `http://localhost/phpflash/listado.php`) y modifique los datos de la tabla navegando por los distintos documentos. Cada vez que realice un cambio (inserción, borrado, modificación), compruebe que éste se hayan producido también en el documento `directorio.xml` (abra otra ventana de

navegador y actualice los contenidos cada vez que se produzcan variaciones en los contenidos de la tabla).

Ahora abra el documento `xml_file fla`, en cuyo primer fotograma de la capa *acciones* encontrará el siguiente código, que es idéntico al del ejercicio anterior, con la salvedad de que la llamada que realiza el objeto XML `datos_bd` se realiza a un documento XML y no a un PHP, como ocurría anteriormente:

```
// declaración de variables a utilizar
inicioX=45;
inicioY=125;
separacionVertical=30;

XML.prototype.ignoreWhite=true;

// función que mostrará los detalles de cada usuario en el campo de texto de la
derecha
mostrarDetalle=function(id){
    var datos_usuario=datos_bd.firstChild.childNodes[id];
    detalles.htmlText="";
    detalles.htmlText+="id: " +datos_usuario.attributes.id+"<br>";
    detalles.htmlText+="nombre: " +datos_usuario.attributes.nombre +
" +datos_usuario.attributes.apellido+"<br>";
    detalles.htmlText+="nick: " +datos_usuario.attributes.nick+"<br>";
    detalles.htmlText+="email: " +datos_usuario.attributes.email+"<br>";
    detalles.htmlText+="url: " +datos_usuario.attributes.url+"<br>";
    detalles.htmlText+="target: " +datos_usuario.attributes.target+"<br>";
}

// objeto LoadVars que recibe el listado de usuarios
```

```

datos_bd=new XML();

// función que procesa la información del listado de usuarios
datos_bd.onLoad=function(exito){
    if(exito){
        var raiz=this.firstChild;
        var hijos=this.firstChild.childNodes.length;
        for(var n=0;n<hijos;n++){
            var
nom=_root.attachMovie("nombre","nombre"+n,n+1,{_x:inicioX,_y:inicioY+(separacionVertical*n)});
            nom.nick.text=raiz.childNodes[n].attributes.nick;
            nom.id=n;
            nom.onRelease = function (){
                this._parent.mostrarDetalle(this.id);
            }
        }
        detalles.htmlText="Elija un usuario en el menú de la izquierda.";
    }else{
        detalles.htmlText="<b>Error al cargar los datos</b>";
    }
}

// acceso al documento XML que contiene el listado de usuarios
datos_bd.load("http://localhost/phpflash/directorio.xml?rn=" + new
Date().getTime());

```

Abra el documento `xml_file.html` en su navegador tecleando `http://localhost/phpflash/xml_file.html`, para observar cómo se ha generado el listado de usuarios de igual manera que en los ejercicios anteriores (figura 6.11)

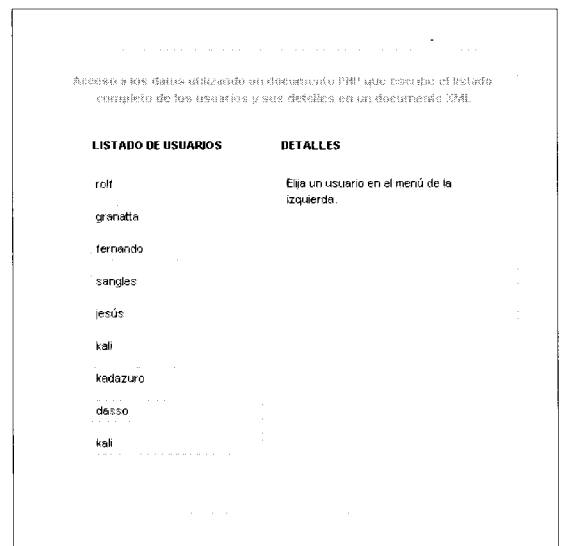


Figura 6.11.

Listado de usuarios extraído del documento `directorio.xml`.

Si pulsa sobre uno de ellos para observar sus detalles, se accede directamente a los datos obtenidos por la instancia del clip de película al recorrer la información del árbol XML (figura 6.12).

```
gotoAndPlay("capitulo_7");
```

Accede a los datos utilizando el clip de película. De esta manera se muestran los detalles de los usuarios, para detalles de cada uno de ellos.

LISTADO DE USUARIOS	DETALLES
rolf	id: 6
granatta	nombre: Kali Romiglia
fernando	nick: kali
	email: kali@romiglia.com
	url: http://www.romiglia.com
sangle:	
jesus:	
kali	
kadzuo	
dasso	
kali	

Figura 6.12.
Detalles del usuario kali.

Capítulo 7

Un caso práctico

Como complemento práctico a los conocimientos adquiridos anteriormente dedicaremos este capítulo a la creación de una agenda cuyos contenidos se gestionan mediante un panel de administrador en el que se podrán insertar, modificar y eliminar registros, a la vez que se mantiene actualizado un documento XML con la información. Además, la agenda contará con la posibilidad de mostrar imágenes de forma dinámica (documentos JPEG), disponiendo el panel de administrador de un formulario de ingreso de datos mediante el cual también subiremos imágenes al servidor mientras que guardamos la ruta de las mismas en la base de datos para su acceso posterior, con lo que podremos añadir nuevas imágenes para las distintas noticias de la agenda o bien elegir una de las que ya hayamos subido mediante una pequeña galería que también vamos a construir.

La naturaleza privada de este panel de administrador hace que el acceso a sus páginas haya de ser restringido; utilizaremos un nombre de usuario y una contraseña para acceder al interior del panel y, una vez dentro, variables en el servidor o variables de sesión para que sólo usuarios "permitidos" tengan la potestad de modificar los contenidos de la agenda.

Para familiarizarnos con estas nuevas funciones, que aplicaremos posteriormente en el caso práctico, comenzaremos por realizar ejemplos simples de cómo subir documentos (en este caso, JPEG) al servidor, cómo realizar una galería de imágenes y, por último, cómo crear secciones de acceso restringido.

Como datos para la agenda que vamos a construir, hemos utilizado las noticias de que dispone *Carlos* en su *blog* personal (<http://www.kadazuro.com/blog>), al que frecuente-

mente añade nuevos trucos y reflexiones sobre su trabajo diario. Cada una de estas noticias, que puede o no disponer de una imagen que la complementa, será imprimible, y además añadiremos un pequeño formulario de contacto para ofrecer a los visitantes de la agenda la posibilidad de enviar sus opiniones.

Creación de aplicación para subir ficheros al servidor

El proceso para poder subir documentos (imágenes JPEG, en nuestro caso) al servidor de que disponemos es relativamente sencillo en PHP. En el formulario que utilizaremos para ello, hemos de crear un campo oculto de nombre `MAX_FILE_SIZE`, en el que especificaremos el tamaño máximo en bytes que vamos a permitir para que el documento pueda ser subido al servidor, además de un campo de texto de tipo fichero (`FILE`).



Nota: Es importante hacer notar que el formulario ha de disponer de formato multiparte, lo que significa que se está especificando que, a la hora de enviar los contenidos del formulario, se van a enviar tanto datos como documentos "adjuntos".

En nuestro caso, estamos interesados en que se puedan subir imágenes al servidor, concretamente en el formato JPG/JPEG, ya que (por ahora) éste es el formato de imagen que Flash puede importar de forma dinámica. A continuación realizaremos un pequeño documento

PHP de ejemplo para subir documentos de imagen al servidor. Dicho documento consta de dos partes: por un lado dispone de un formulario con el campo de texto de tipo fichero (`FILE`), y por otro dispone del fragmento de código que almacena una copia del fichero de imagen en el servidor.

La etiqueta del formulario multiparte se declara de la siguiente forma:

```
<form enctype="multipart/form-data"
action="<? echo $PHP_SELF;?>"
method="post">
```

Mediante la variable `enctype` se especifica el envío tanto de valores de variables como de documentos. Posteriormente, añadimos un campo oculto con el valor del tamaño máximo en bytes que permitiremos para los ficheros que vamos a grabar en el servidor:

```
<input type="hidden"
name="MAX_FILE_SIZE" value="100000">
```

Si omitimos la declaración de este campo, el tamaño máximo permitido es el que se especifica en el documento de configuración de PHP, que, por lo general, suele ser de 8 Mb (8192 bytes). El configurar este campo de texto nos permite tener control sobre el tamaño de los ficheros para así disminuir el tiempo de carga de los mismos.

A continuación, introducimos el campo de texto de tipo fichero:

```
Subir esta imagen: <input
name="imagen" type="file">
```

La variable que contendrá el documento utilizará el nombre imagen (`name="imagen"`). Por último, añadimos el botón de envío y cerramos la etiqueta del formulario:

```
<input type="submit" name='submit'
value="Subir Fichero">
</form>
```

Cuando enviamos documentos mediante el método POST utilizando un formulario multiparte, PHP guarda la información en una variable global llamada `$_FILES` o `$HTTP_POST_FILES` (esta última era utilizada en la versión 4.1.0 de PHP, y permanece disponible por motivos de compatibilidad, pero ambas contienen la misma información), un *array* de tipo lista, en el que utilizaremos como apuntador el nombre del campo especificado en el formulario (*imagen* en nuestro caso) para poder manipular la información del fichero. Por lo general, las variables son las siguientes:

- `$_FILES['imagen']['name']` almacena el nombre original del documento en la máquina cliente.
- `$_FILES['imagen']['type']` almacena el tipo MIME del documento (si el navegador lo proporciona), por ejemplo, "image/gif".

- `$_FILES['imagen']['size']` almacena el tamaño en bytes del documento recibido.
- `$_FILES['imagen']['tmp_name']` almacena el nombre del documento temporal que se utiliza para grabar en el servidor el archivo recibido. Cuando PHP va a almacenar un documento, éste es guardado en la carpeta configurada para movimientos temporales (PHP/temp o PHP/uploadtemp), y se le asigna un nombre único temporal, para ser eliminado al terminarse la ejecución del documento PHP. Antes de que esto ocurra, debemos mover el fichero recién subido a la carpeta que deseemos, asignándole un nombre que nos sea de utilidad.

Cuando creamos el código PHP que almacenará la imagen en el servidor, hemos de revisar primeramente si la variable `submit` existe (para comprobar si el formulario ha sido enviado):

```
<?php
if(isset($submit)){
```

A continuación, comprobamos si el documento ha sido grabado en la carpeta temporal y si es un archivo que ha llegado ahí mediante el envío del formulario, para lo que usamos la función `is_uploaded_file`, cuya sintaxis es:

```
is_uploaded_file(nombre temporal de fichero)
```

Esta función retorna `true` si el documento ha sido subido con éxito a la carpeta de documentos temporales:

```
if (is_uploaded_file($_FILES['imagen']['tmp_name'])) {
```

Si la imagen es del tipo JPG o JPEG, le asignamos un nuevo nombre, que será obtenido mediante el uso de la función `time` (que retorna la cantidad de segundos transcurridos desde las 0 horas del 1 de enero de 1970) para asegurarnos de que cada nuevo documento que se sube al servidor dispone de un nombre único, puesto que éste cambiará cada segundo que pasa:

```
if ($_FILES['imagen']['type'] == "image/jpeg" || $_FILES['imagen']['type'] ==
"image/pjpeg"){
    $nuevoNombre = time().".jpg";
```



Nota: Observe que a la variable que contiene el nombre (`$nuevoNombre`) le hemos agregado un punto y la extensión `".jpg"`.

Para mover el documento a una carpeta específica, podemos especificar la ruta absoluta en la que queremos almacenarla, o bien podemos utilizar la ruta relativa al documento PHP en el que estamos trabajando. Si decide elegir este último modo, guarde el documento que estamos realizando con el nombre de `upload_jpg.php` en la raíz del servidor (carpeta `www.ohdocs`, según su configuración) y posteriormente cree allí una carpeta con el nombre `fotografias`. A continuación, vamos a copiar el fichero recién subido a esta carpeta. Podemos hacerlo de dos formas:

```
copy(nombre temporal, nombre final);
```

O bien:

```
move_uploaded_file(nombre temporal,
nombre final);
```

Puede elegir la que prefiera; en este ejemplo utilizaremos la segunda:

```
move_uploaded_file($_FILES['imagen']
['tmp_name'], "fotografias/
$nuevoNombre ");
```

El nombre que PHP asigna al documento cuando lo graba en la carpeta temporal (`$_FILES['imagen']['tmp_name']`) es el que utilizamos para moverlo, junto con la carpeta donde queremos que lo guarde (`"fotografias/$nuevoNombre"`). Para obtener la información del fichero una vez que está en el servidor y desplegarla junto con un mensaje de que se ha realizado correctamente la operación, podemos hacer uso de la función `GetImageSize`, cuya sintaxis es:

```
GetImageSize("directorio/nombre_imagen");
```

Y que retorna un *array* de cuatro campos:

- El campo 0 contiene la anchura en píxeles de la imagen.
- El campo 1 contiene la altura en píxeles de la imagen.

- El campo 2 almacena un tipo numérico, cuyo valor es asociado a un tipo de imagen; 1 corresponde a documentos GIF, 2 corresponde a documentos JPG y 3 corresponde a documentos PNG.
- El campo 3 almacena una cadena de texto en formato HTML con la anchura y la altura de la imagen ("width=xxx height=xxx").

Almacenamos la información que retorna la función en la variable \$data:

```
$data = GetImageSize("fotografias/
$nuevoNombre");
```

Y mostramos el mensaje de éxito de la operación junto con la imagen, utilizando la ruta hasta la misma y la cadena de texto que corresponde a su altura y su altura en formato HTML:

```
echo "<img src='fotografias/
$nuevoNombre' $data[3]> <br> imagen
$nuevoNombre subida con éxito";
```

En caso de que la imagen enviada no sea del tipo JPG/JPEG, mostramos un mensaje alternativo:

```
}else{
    echo "Formato no válido
    para fichero de imagen";
}
```

Si se produce un error a la hora de subir la imagen o bien que haya sobrepasado el tamaño permitido en bytes, lo notificamos también con un mensaje:

```
} else {
    echo "Error al cargar
    imagen: ".$_FILES['imagen']['name'];
}
```

Por último cerramos tanto la sentencia if en la que comprobábamos la existencia de la variable submit como el fragmento de código PHP:

```
}
?>
```

El código completo (del que puede encontrar una copia en el documento material/cap7/php/upload_jpg.php del CD) es el siguiente:

```
<?php
if(isset($submit)){
    if (is_uploaded_file($_FILES['imagen']['tmp_name'])) {
        //revisamos que sea jpg
        if ($_FILES['imagen']['type'] == "image/jpeg" || $_FILES['imagen']['type']
        == "image/pjpeg"){
            //nuevo nombre para la imagen
            $nuevoNombre = time()."jpg";
            //movemos la imagen
            move_uploaded_file($_FILES['imagen']['tmp_name'], "fotografias/
            $nuevoNombre");
            //obtenemos la información
            $data = GetImageSize("fotografias/$nuevoNombre");
            //mensaje de éxito
            echo "<img src='fotografias/$nuevoNombre' $data[3]> <br> imagen
            $nuevoNombre subida con éxito";
        }else{
            echo "Formato no válido para fichero de imagen";
        }
    } else {
```

```

    echo "Error al cargar imagen: " . $_FILES['imagen']['name'];
}
}
?>
<form enctype="multipart/form-data" action="<? echo $PHP_SELF;?>"
method="post">
<input type="hidden" name="MAX_FILE_SIZE" value="100000">
Subir esta imagen: <input name="imagen" type="file">
<input type="submit" name='submit' value="Subir Fichero">
</form>

```

Una vez que se ha asegurado de que dispone de la carpeta fotografías en la raíz de su servidor local, abra el documento PHP en su navegador, tecleando `http://localhost/upload_jpg.php` y elija un documento JPEG del que disponga en su disco duro. Al pulsar el botón **Subir fichero**, si todo se ha ejecutado correctamente, obtendrá un mensaje de éxito junto con la foto que acaba de almacenar en el servidor, tal y como puede observar en la figura 7.1. Si accede a la carpeta fotografías de su servidor local, encontrará la imagen con el nombre con el que se ha almacenado junto con la extensión .jpg.

Creación de una galería de imágenes

La galería de imágenes es aquella sección en la que vamos a poder visualizar las fotografías que hemos subido al servidor y que se han almacenado en la carpeta fotografías. Para ello, abriremos este directorio e iremos leyendo cada uno de los documentos que contiene, a los que no haremos verificación de tipo, pues partimos de la base de que todos ellos son imágenes.

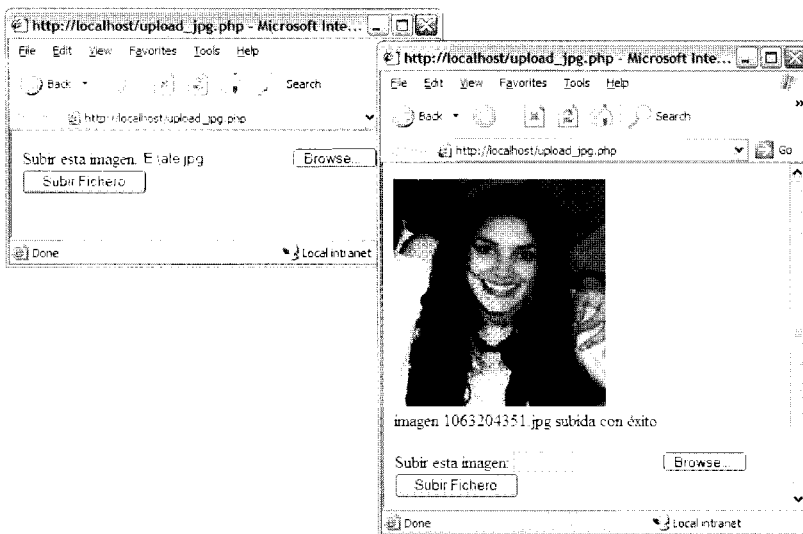


Figura 7.1.
Formulario para subir la imagen al servidor y pantalla de éxito de la operación.

Cada directorio contiene dos archivos que son enlaces a la carpeta padre del directorio (..) y a la carpeta raíz (.), siendo ésta la única verificación que habremos de realizar.

Para mostrar los documentos almacenados en el directorio utilizaremos un bucle, creando una tabla con tres celdas por fila; para saber cuándo llegamos a la tercera celda utilizaremos un contador que se va incrementando en cada iteración del bucle, con lo que si el valor es igual a 3, hemos de cerrar la fila actual y crear una nueva (mediante las etiquetas HTML `<tr>y</tr>`). Comenzaremos abriendo el directorio:

```
<?
$handle=opendir("fotografias");
Iniciamos el contador de celdas para
la tabla.
$count=0;
?>
```

Tras este fragmento de código PHP, creamos una tabla junto con la etiqueta de su primera fila, en lenguaje HTML:

```
<table width="400" border="0"
cellspacing="0" cellpadding="10"
align="center">
<tr>
```

A continuación, y de nuevo mediante código PHP, creamos un bucle que se ejecutará mientras existan documentos en la carpeta que estamos leyendo:

```
<?
while ($file = readdir($handle)) {
```

La función `readdir` retorna el nombre del siguiente documento existente en el directorio, que es guardado en la variable `$file`, en la que comprobamos que el valor almacenado no sean la carpeta padre o raíz:

```
if ($file != "." && $file != "..") {
```

En caso de que así sea, obtenemos la ruta completa al fichero para así poder evaluarlo:

```
$fichero = "fotografias/".$file;
```

Con objeto de verlas ordenadas, en la galería visualizaremos las distintas imágenes a un tamaño de 100x100 píxeles; sin embargo, podemos mostrar la información de las dimensiones reales de los documentos del directorio, utilizando la función `GetImageSize` y guardando el resultado en la variable `$fileData`:

```
$fileData = GetImageSize($fichero);
```

Al disponer de tres celdas por fila, cada una de ellas utilizará una tercera parte de la anchura de la tabla (33 por ciento). En las distintas celdas de la misma introduciremos las imágenes, junto con su nombre y dimensiones:

```
echo "<td align='center' width
='33%' ><img src='$fichero'
width=100 height=100 border=0><br>
$file <br>( $fileData[0] x
$fileData[1] )</td>\n";
```

Crearemos la etiqueta de la celda con la alineación en el centro y un ancho del 33 por ciento mediante:

```
<td align='center' width ='33%' >
```

Establecemos posteriormente la ruta hacia la imagen (almacenada en la variable `$fichero`), junto con su altura y anchura, mostrando a continuación el nombre del documento y sus atributos, utilizando el formato (ancho x alto):

```
<img src='$fichero' width=100
height=100 >
<br> $file <br>
( $fileData[0] x $fileData[1] )
```

Terminamos la impresión cerrando la etiqueta de la celda y posteriormente incrementamos la variable que almacena el número de celdas de cada fila:

```
</td>\n";
$count++;
```

Si el valor almacenado en la variable `$count` es 3, cerramos la fila y abrimos una

nueva, inicializando a cero el valor del contador:

```
if($count==3){
    $count = 0;
    echo "</tr><tr>";
}
```

Cerramos el bucle `while` y el directorio que estamos leyendo:

```
}
closedir($handle);
```

Salimos de PHP y cerramos la tabla con sus correspondientes etiquetas HTML:

```
?>
</tr>
</table>
```

El código completo (del que puede encontrar una copia en el documento `material/cap7/php/galeria.php` del CD) para el documento sería el siguiente:

```
<?
//abrimos el directorio.
$handle=opendir("fotografias");
//contador para filas y columnas
$count=0;
?>
<table width="400" border="0" cellspacing="0" cellpadding="10" align="center">
<tr>
<?
while ($file = readdir($handle)) {
    if ($file != "." && $file != "..") {
        $fichero = "fotografias/".$file;
        $fileData = GetImageSize($fichero);
        echo "<td align='center' width ='33%' ><img src='$fichero' width=100
height=100><br> $file <br>( $fileData[0] x $fileData[1] )</td>\n";
        $count++;
        if($count==3){
            //cerramos la fila.
            $count = 0;
            echo "</tr><tr>";
        }
    }
}
//cerramos el directorio.
closedir($handle);
?>
</tr>
</table>
```

Cree un nuevo documento de texto y añada este código, para posteriormente guardarlo con el nombre de `galeria.php` en la carpeta raíz de su servidor local. Ábralo entonces en su navegador tecleando `http://localhost/galeria.php`, y podrá observar cómo se crea una tabla que muestra, con tres celdas por fila, las imágenes contenidas en el directorio `fotografias`, tal y como se muestra en la figura 7.2. Pruebe a añadir nuevas imágenes a dicho directorio utilizando el documento que creamos en el apartado anterior (`upload_jpg.php`) y vuelva a abrir el documento `galeria.php` en su navegador para observar cómo cambia la galería de imágenes.

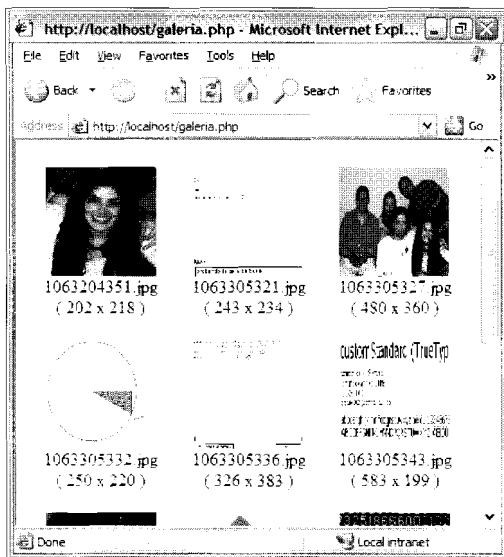


Figura 7.2.
Galería de imágenes mostrando los archivos de una carpeta.

Creación de zonas de acceso restringido

Para crear zonas de acceso restringido en nuestros proyectos podemos optar por proteger los directorios del servidor directamente, usando *cookies* o variables en el servidor (variables de sesión). Mientras que las *cookies* son variables que se almacenan en el ordenador del usuario, las sesiones se guardan en el servidor, con la ventaja de que son eliminadas cuando el navegador se cierra y además no almacenan ninguna información en el ordenador del usuario, lo que nos proporciona la seguridad de que esos datos no serán accedidos en el futuro.

Para generar sesiones, debemos inicializarlas en el documento en el que se vayan a utilizar, para lo que ejecutamos la función `session_start`:

```
session_start();
```

Una vez iniciada la sesión, se crea un *array* de lista de nombre `$_SESSION` en el que se almacenan variables que son accedidas de la siguiente forma:

```
$_SESSION['nombre'] = valor;
```

Para eliminar valores de una sesión, se suele utilizar la función `unset(nombre_variable)`:

```
unset($_SESSION['nombre']);
```

Si desea eliminar completamente la sesión, puede hacer uso de la función `session_destroy`:

```
session_destroy();
```

Para ver un ejemplo simple del funcionamiento de las sesiones, abra un nuevo documento de texto y copie el siguiente código (disponible en el documento `material/cap7/php/sesiones1.php`), para posteriormente guardar el documento con el nombre `sesiones1.php` en la raíz de su servidor:

```
<?php
session_start();
if (!isset($_SESSION['contador'])) {
    $_SESSION['contador'] = 0;
} else {
    $_SESSION['contador']++;
}
echo $_SESSION['contador'];
?>
```

Lo que se espera que haga este documento cuando lo visualice en su navegador tecleando `http://localhost/sesiones1.php` es lo siguiente: cuando lo abra por primera vez, el documento almacenará el valor 0, es decir, la variable de sesión de nombre `contador` no existe, y la inicializamos asignándole el valor 0. Cuando actualice este documento (pulsando el botón **Actualizar** o la tecla **F5**), la variable, puesto que ya existe, incrementará su valor en una unidad. Con cada actualización del documento en su navegador podrá observar cómo el contenido de la variable `contador` se va incrementando al tomar los datos de la variable de sesión del servidor, tal y como se muestra en la figura 7.3.

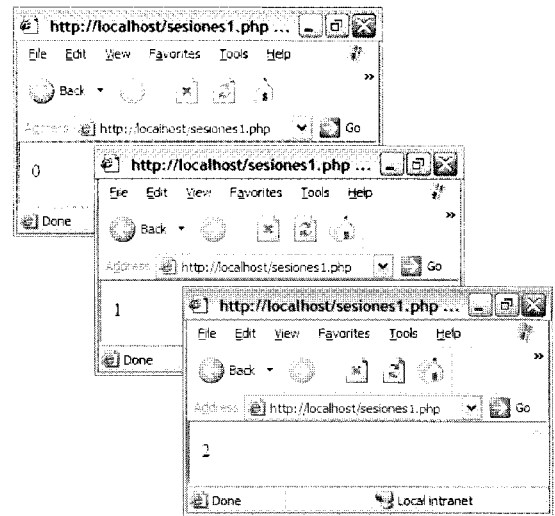


Figura 7.3.

Variables de sesión cambiando de valor.

La zona de acceso restringido la construiremos basándonos en el funcionamiento de un módulo realizado con este propósito: `PHPSecurePages`. Puede descargar dicho módulo de `http://www.phpsecurepages.com`, aunque en nuestro caso trabajaremos con una versión simplificada y exclusiva para versiones de PHP superiores a la 4.0.6, siendo su funcionamiento de la siguiente forma:

Cada uno de los documentos incluye un solo archivo (`secure.php`) que dispone de dos opciones: dar ingreso a los usuarios (`login`) o darles salida (`logout`). Dependiendo del caso, se realiza la inclusión de un documento o de otro, incluyendo `checkLogin.php`, si se desea dar ingreso, y `logout.php`, si se desea dar salida.

Cada vez que se acceda a una nueva página, `checkLogin.php` comprueba si existen las variables de sesión `usuario` y `clave`. Si no existen, revisa si se ha enviado el formulario

donde se solicitaron estos datos deteniendo la ejecución de la página si las variables del formulario tampoco existen, incluyendo antes el formulario (`interface.php`) para imposibilitar el que se pueda ver la página. Si las variables existen, ya sea vía `POST` o vía `SESSION`, se comparan dichos datos con los almacenados en la tabla de usuarios de la base de datos.

Para comprobar el funcionamiento de este módulo necesitará crear en su base de datos una tabla con la información de los usuarios. Abra el administrador de la base de datos (`http://localhost/phpMyAdmin`), seleccione la base de datos `pruebas` y presione la pestaña `SQL`. Una vez allí, localice el documento `usuarios.sql` (que puede encontrar en la carpeta `material/cap7/sql` del CD) y copie los datos en el área de texto (o bien pulse sobre el botón **Buscar** bajo la opción **Localizar archivo de texto** para recorrer los directorios hasta la dirección en la que se encuentre el archivo). Al presionar el botón **Continuar** se creará la tabla `usuarios` en la base de datos `pruebas`. Esta tabla dispone de tres campos: `id`, `usuario` y `clave`. Observe cómo el documento `usuario.sql`, además de crear la tabla,

también realiza el ingreso de un registro: el usuario `admin`, cuya contraseña es `"admin"`; este usuario será el que utilizaremos tanto para esta práctica como para acceder al Panel de Administrador.

Como necesitaremos comparar la información de la base de datos con la ingresada mediante el formulario o bien guardada en sesiones, haremos uso de los archivos de configuración (`config.php` y `funciones.php`) que hemos venido utilizando a lo largo de los últimos capítulos.



Nota: Recuerde incluir los dos archivos de configuración antes de incluir el documento `secure.php`.

A continuación analizaremos el comportamiento del documento `secure.php` (que puede encontrar en la carpeta `material/cap7/php/admin` del CD), que revisa la existencia de la variable `logout` (línea 3). Si la variable existe, incluye el documento `logout.php` (línea 5). En caso contrario, incluye el otro documento, `checkLogin.php` (línea 8):

```
1  <?php
2  // login o logout?
3  if (isset($logout) || isset($_GET["logout"]) || isset($_POST["logout"])) {
4      // logout
5      include("logout.php");
6  } else {
7      // login
8      include("checkLogin.php");
9  }
10  ?>
```

El código del documento `checkLogin.php` (que puede encontrar en la carpeta `material/cap7/php/admin` del CD) es el siguiente:

```
1  <?php
2  // revisamos si es login por sesiones o por formulario
3  if (!isset($_POST['usuario_digitado']) && !isset($_POST['clave_digitada'])) {
4      session_start();
5      //usamos los valores de las sesiones
6      $usuario = $_SESSION['usuario'];
7      $clave = $_SESSION['clave'];
8  }else{
9      // usamos los datos ingresados
10     session_start();
11     //borramos las sesiones por si existen
12     unset($_SESSION['usuario']);
13     unset($_SESSION['clave']);
14
15     $usuario = $_POST['usuario_digitado'];
16     $clave = $_POST['clave_digitada'];
17     $_SESSION['usuario'] = $usuario;
18     $_SESSION['clave'] = $clave;
19 }
20
21 if (!$usuario) {
22     // no hay login disponible
23     include("interface.php");
24     exit;
25 }
26 if (!$clave) {
27     // no hay contraseña
28     $mensaje = "contraseña incorrecta";
29     include("interface.php");
30     exit;
31 }
32 // nos conectamos a la bd
33 $cnx = conectar();
34 //buscamos al usuario
35 $userQuery = mysql_query("SELECT * FROM usuarios WHERE usuario =
'$usuario'") or die(mysql_error());
36 // revisamos usuario y password
37 if (mysql_num_rows($userQuery) > 0) {
38     // usuario existe, seguimos
39     $userArray = mysql_fetch_array($userQuery);
40
41     if ($usuario != $userArray['usuario']) {
42         // caso sensitivo, usuario no está presente en bd
43         $message = "Usuario no Existe";
44         echo $message;
45         include("interface.php");
46         exit;
47     }
48     if (!$userArray['clave']) {
49         // no tiene clave en bd, no entra
50         $message = "No se encontró contraseña para el usuario";
```

```

51             include("interface.php");
52             exit;
53         }
54         if (stripslashes($userArray['clave']) != $clave) {
55             // contraseña es incorrecta
56             $message = "Contraseña es incorrecta";
57             include("interface.php");
58             exit;
59         }
60     }else{
61         // usuario no existe del todo.
62         $message = "Usuario no Existe";
63         include("interface.php");
64         exit;
65     }
66     //si hemos llegado hasta aqui significa que el login es correcto.
67     ?>

```

El formulario de ingreso del documento `interface.php` contiene dos campos de texto: `usuario_digitado` y `clave_digitada`. En la línea 2 podemos observar cómo se comprueba la existencia vía POST de estas dos variables. Si no existen, se crea una nueva sesión (línea 4) y se asigna la información almacenada en la sesión a las variables `$usuario` y `$clave` (líneas 6 y 7). Si estas variables ya existen, también se inicia la sesión (línea 10), pero en este caso guardaremos en ella los datos enviados mediante el formulario, de modo que cuando el usuario acceda a la siguiente página, la comprobación de los datos no se realice en base al formulario, sino a los datos de la sesión. Para ello, eliminamos las variables de sesión `$usuario` y `$clave` (en caso de que existan, líneas 12 y 13) y almacenamos los valores enviados mediante el formulario en las variables `$usuario` y `$clave` (líneas 15 y 16) para posteriormente asignar éstos a las variables de sesión (líneas 17 y 18).

Posteriormente, comprobamos la existencia de la variable `$usuario` (línea 21), incluimos el documento `interface.php` (línea 23), que es el que contiene el formulario, y finalmente terminamos la ejecución de la página (línea 24).

Las líneas 26-31 realizan la misma tarea que las anteriores cuando comprueban la variable `$usuario`, pero comprobando en esta ocasión la existencia de la variable `$clave`.



Nota: Cada vez que ocurra un error es incluido el documento `interface.php` y se termina la ejecución de la página, no permitiendo así mostrar el documento al que se quiere acceder, sino el formulario en el que se solicita el ingreso de los datos.

Cuando hemos comprobado positivamente la existencia de las variables, necesitamos realizar la comparación con la información almacenada en la base de datos, por lo que establecemos una conexión con ésta (línea 33) y realizamos la consulta SQL en la que se seleccionan los campos correspondientes al registro cuyos contenidos de la columna *usuario* coincide con el contenido de la variable `$usuario` (línea 35); almacenamos el resultado de la consulta en la variable `$userQuery` y comprobamos si existen filas en dicha variable (línea 37) para acceder a la información. En caso de ser así, creamos la variable `$userArray` con la información que obtenemos al utilizar la función `mysql_fetch_array()`, para a continuación comparar el valor contenido en `$usuario` con el resultado de la consulta a la base de datos (líneas 41-47). Luego habremos de comprobar si el usuario en cuestión dispone de contraseña (líneas 48-53) y, finalmente, que la contraseña de que disponga coincida con la almacenada en la base de datos (líneas 54-59).

Si la consulta SQL no retorna filas, el usuario no existe en la base de datos, por lo que ha de abandonar el sistema (líneas 60-65).

Si se produce fallo y desea informar al usuario de qué error se produjo cuando intentó acceder al sistema, creamos una variable `$message` en aquellos lugares en los que puede producirse error, que podremos mostrar posteriormente cuando incluyamos de nuevo el documento `interface.php`.

El documento `logout.php` se encarga de eliminar las sesiones de la mejor forma posible; primero inicializa la sesión (línea 2), posteriormente borra el valor de las variables de sesión `usuario` y `clave` (líneas 3 y 4), elimina todas las demás sesiones reiniciando la variable `$_SESSION` al asignarle como valor un *array* vacío (línea 5), destruye la sesión utilizando la función `session_destroy` (línea 6) y, por último, elimina la *cookie* que almacena el nombre de la sesión (que por lo general suele ser `PHPSESSID`).

El código de este documento (que puede encontrar en la carpeta `material/cap7/php/admin` del CD) es el siguiente:

```
1  <?PHP
2  session_start();
3  unset($_SESSION['usuario']);
4  unset($_SESSION['clave']);
5  $_SESSION = array();
6  session_destroy();
7  $sessionPath = session_get_cookie_params();
8  setcookie(session_name(), "", 0, $sessionPath["path"], $sessionPath["domain"]);
9  ?>
```

El documento `interface.php` contiene el formulario que se muestra si el usuario no ha ingresado en el sistema y cuya actividad consiste en enviar la información a la misma página que está tratando de verse (por lo que se utiliza la variable

`$_SERVER['PHP_SELF']`), con la diferencia de que en este último caso la variable de servidor `PHP_SELF` retorna la ruta del archivo pero no las variables, es decir, si está ingresando al documento `index.php?id=5`, `PHP_SELF` devolverá como ruta únicamente `index.php`, obviando las variables que se muestran a continuación del carácter `?`. Para obtener estas variables, almacenamos la ruta hasta el archivo en la variable `$documentLocation` (línea 2) y comprobamos la existencia de la variable de servidor `$QUERY_STRING` para saber si existen más variables (línea 3). Si existe dicha variable, la adjuntamos a la ruta del documento (línea 4).

Las líneas 9-32 contienen la declaración de un fragmento de código escrito en JavaScript que verifica la validez de la información del formulario antes de que sea enviada.

La línea 36 muestra la declaración del formulario; observe cómo se ha sustituido la variable `PHP_SELF`, que hemos utilizado en algunos ejemplos anteriormente, por la variable `$documentLocation`.

En caso de que se produzca un fallo al ingresar el usuario y exista un mensaje de error (almacenado en la variable `$message` del documento `checkLogin.php`), se imprime éste (líneas 43-47).

En las líneas 56 y 60 se encuentran los campos de introducción de texto `usuario_digitado` y `clave_digitada`, respectivamente. El botón de envío del formulario se encuentra en la línea 64.

El código de este documento (que puede encontrar en la carpeta `material/cap7/php/admin del CD`) es el siguiente:

```
1  <?
2  $documentLocation = $_SERVER['PHP_SELF'];
3  if ( $_SERVER['QUERY_STRING'] ) {
4  $documentLocation .= "?" . $_SERVER['QUERY_STRING'];
5  }
6  ?>
7  <html><head>
8  <title></title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
9  <SCRIPT LANGUAGE="JavaScript">
10 <!--
11 function checkData() {
12     var fl = document.forms[0];
13     var wm = "Ocurrieron los siguientes Errores :\n\r\n";
14     var noerror = 1;
15     var tl = fl.usuario_digitado;
16     if (tl.value == "" || tl.value == " ") {
17         wm += "Digite su nombre de Usuario\r\n";
18         noerror = 0;
19     }
20     var tl = fl.clave_digitada;
21     if (tl.value == "" || tl.value == " ") {
```

```

22     wm += "Digite la contraseña\r\n";
23     noerror = 0;
24 }
25 if (noerror == 0) {
26     alert(wm);
27     return false;
28 }
29 else return true;
30 }
31 //-->
32 </SCRIPT>
33 </head>
34
35 <body onLoad="document.form1.usuario_digitado.focus()" >
36 <form name="form1"action='<?PHP echo $documentLocation?>' METHOD="post"
onSubmit="return checkData()">
37     <table width="500" cellpadding="0" cellspacing="0" >
38         <tr>
39             <td height="30" COLSPAN="2" ALIGN="left">Ingreso de Usuarios</td>
40         </tr>
41         <tr>
42             <td height="24" COLSPAN="2" ALIGN="center"> <i><NOBR>
43 <?PHP
44 // revisa si hay mensajes de error.
45 if ($message) {
46     echo $message;
47 } ?>
48 </NOBR></i>
49 </td>
50 </tr>
51 <tr>
52     <td ALIGN="right" valign="bottom">
53         <table width="500" cellpadding=4 cellspacing=1 >
54             <tr>
55                 <td align="right">usuario: </td>
56                 <td> <input type="text" name="usuario_digitado" ></td>
57             </tr>
58             <tr>
59                 <td align="right">contraseña: </td>
60                 <td> <input type="password" name="clave_digitada"></td>
61             </tr>
62             <tr>
63                 <td>&nbsp;&nbsp;&nbsp;</td>
64                 <td align="right"><input name="Submit" type="submit" value="Submit"></
td>
65             </tr>
66         </table>
67     </td>
68 </tr>
69 </table>
70 </form>
71 </body>
72 </html>

```

Para observar el funcionamiento de estas páginas copie la carpeta `admin` (que encontrará en la carpeta `material/cap7/php` del CD) en la raíz del servidor.

Esta carpeta, además de los documentos que hemos explicado, incluye, junto con los de configuración (`config.php` y `funciones.php`), otros dos: `index.php` y `salir.php` (recuerde que para que este ejemplo funcione correctamente ha de haber creado la tabla *usuarios* en la base de datos *pruebas*).

El documento `index.php` es aquel al que se quiere acceder cuando desde el navegador tecleamos `http://localhost/admin`.

Su código (del que puede encontrar una copia en el documento carpeta `material/cap7/php/admin/index.php` del CD) es el siguiente:

```
<?php
include ("config.php");
include ("funciones.php");
include ("secure.php");
?>
ha ingresado a la página del
administrador.<br>
<a href="salir.php">salir</a>
```

Este documento incluye los archivos de configuración junto con `secure.php`, que es el que revisa que el usuario ingrese o no correctamente.

Si accede de forma correcta, observará en su pantalla el texto "ha ingresado a la página del administrador" junto con un enlace al documento `salir.php`.

Si en cambio ha introducido datos erróneos, verá un mensaje en el que se le notifica tal circunstancia (figura 7.4).

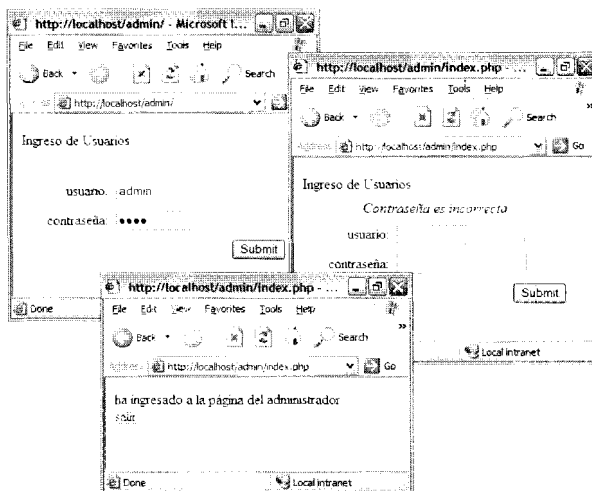


Figura 7.4.

Formulario de ingreso, ingreso erróneo y mensaje de ingreso correcto.

El documento `salir.php` borrará las sesiones y mostrará un mensaje para ingresar de nuevo al sistema con un enlace que apunta al documento `index.php` (figura 7.5), el cual mostrará de nuevo el formulario de ingreso, puesto que las sesiones han sido eliminadas. Su código (del que puede encontrar una copia en el documento carpeta `material/cap7/php/admin/salir.php` del CD) es el siguiente:

```
<?php
include ("config.php");
include ("funciones.php");
$logout = true;
include ("secure.php");
?>
ha salido de la p gina del
administrador.<br>
<a href="index.php"?>ingresar</a>
```

Agenda: Administrador de contenidos + Interfaz

A continuaci n desarrollaremos, tal y como comentamos al inicio de este cap tulo, una

peque a agenda como aplicaci n pr ctica de todo lo aprendido en los cap tulos anteriores, tanto de Flash como de PHP/MySQL; para comenzar, copie la carpeta `material/cap7/proyecto` en la ra z de su servidor local. Desde este momento, iremos explicando cada uno de los documentos que se utilicen, ya sean los completamente nuevos o bien los que presenten cambios con respecto a versiones anteriores ya explicadas en este libro.

Estructura del proyecto

El proyecto est  organizado en carpetas. La carpeta `admin` contiene todos los documentos del m dulo de administrador. Los documentos que incluyen las variables y funciones de uso com n se encuentran en la carpeta `includes`, y en la carpeta `fotografias` encontrar  las im genes que se suben al servidor cuando a ada una nueva noticia.

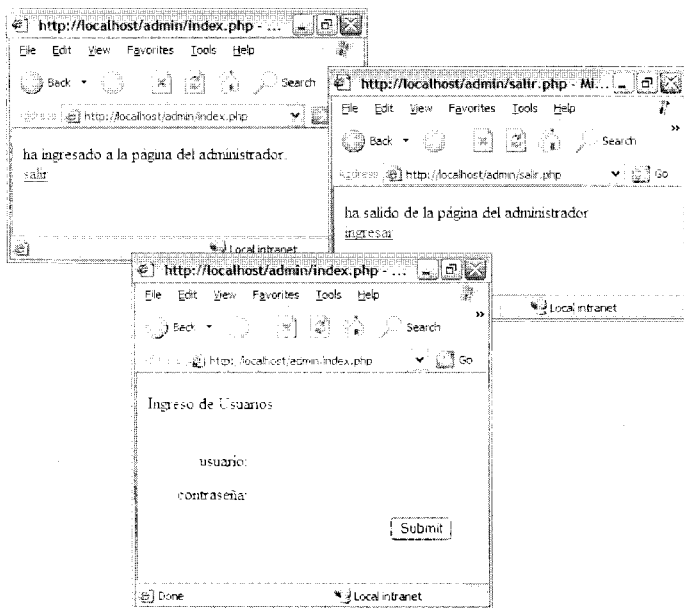


Figura 7.5.

Salida del sistema de administraci n y formulario para volver a acceder

Tabla agenda

Para almacenar nuevas noticias en la agenda, habremos de crear una nueva tabla (de nombre *agenda*) en la base de datos *pruebas*. En esta tabla necesitamos guardar el título de la noticia y el cuerpo de la misma, la fecha, indicar si hay o no una imagen asociada y, por último, un identificador que nos asegure que cada registro será único.

Tabla 7.1.

Estructura de la tabla agenda de la base de datos pruebas.

Nombre del campo	Tipo de datos
id	Entero mediano, autoincremental
cabecera	Varchar de 50 caracteres
texto	Text
fecha	Varchar de 8 caracteres
foto	Varchar de 40 caracteres

En el caso del campo *fecha*, los datos no serán del tipo de fecha de MySQL, ya que Flash usa un registro de meses base cero (*Enero* es el mes 0, *Febrero* es el mes 1, etc.) mientras que MySQL usa un registro de meses base uno (*Enero* es el mes 1, *Febrero* es el mes 2, etc.), dato que hemos

de tener en cuenta para mostrar la fecha correctamente en Flash. Por este motivo, el tipo de datos de esta columna será *varchar* de ocho caracteres (por ejemplo, 20030012 representaría el día 12 del mes 00 -*Enero*- del año 2003: cuatro dígitos para el año, dos para el mes y dos para el día, un formato que sería "aaaammdd").

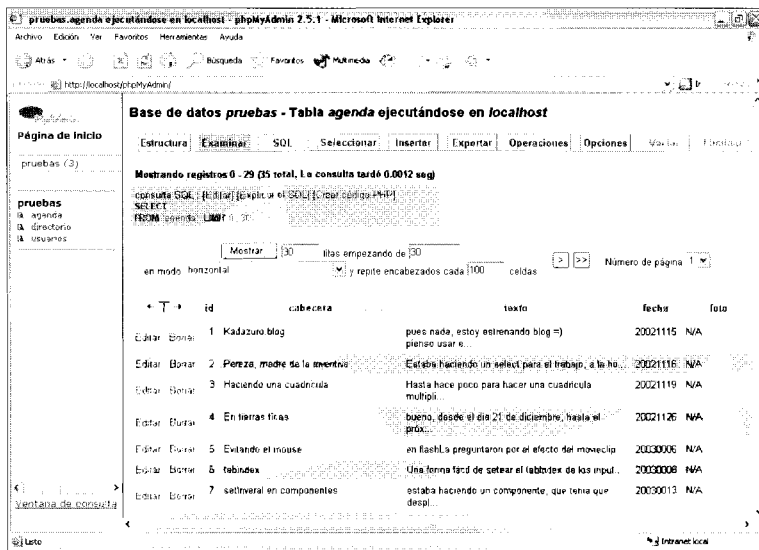
En el campo *foto* se almacenará la ruta de la imagen asociada a la noticia, no la fotografía misma, y los nombres de estas imágenes serán únicos para evitar posibles sobreescrituras accidentales.

Abra el manejador de la base de datos (<http://localhost/phpMyAdmin>), seleccione la base de datos *pruebas* y presione la pestaña SQL. Una vez allí, localice el documento *tabla_agenda.sql* (que puede encontrar en la carpeta *material/cap7/sql* del CD) y copie los datos en el área de texto (o bien pulse sobre el botón **Buscar** bajo la opción **Localizar archivo de texto** para recorrer los directorios hasta la dirección en la que se encuentre el archivo).



Nota: Asegúrese de que, una vez ingresados los contenidos en la tabla *agenda*, dispone en la carpeta *proyecto/fotografias* de su servidor local de una copia de los documentos existentes en la carpeta *material/cap7/proyecto/fotografias*, con objeto de que pueda probar el funcionamiento de la agenda tal y como la hemos desarrollado.

Al presionar el botón **Continuar** se creará la tabla *agenda* en la base de datos *pruebas*. Ahora repita el mismo proceso pero utilizando el documento `datos_agenda.sql` (que puede encontrar en la misma carpeta del CD que el anterior) para almacenar en la tabla recién creada la información con la que hemos trabajado para construir la agenda (figura 7.6).



Base de datos pruebas - Tabla agenda ejecutándose en localhost

Mostrando registros 0 - 29 (35 total, la consulta tardó 0.0012 seg)

consulta SQL: `SELECT * FROM agenda LIMIT 0, 30`

en modo: horizontal 30 filas empezando de 30 y repite encabezados cada 100 celdas Número de página 1

	id	cabecera	texto	fecha	foto
Editar Borrar	1	Kadazuro blog	pues nada, estoy estrenando blog =)	20021115	N/A
Editar Borrar	2	Perez, mode de la inventiva	Estaba haciendo un select para el trabajo, a la ho...	20021116	N/A
Editar Borrar	3	Haciendo una cuadrícula	Hasta hace poco para hacer una cuadrícula multipl...	20021119	N/A
Editar Borrar	4	En tierras frías	bueno, desde el día 21 de diciembre, hasta el próx...	20021126	N/A
Editar Borrar	5	Evitando el mouse	en flash la prepararon por el efecto del mouseclic	20030006	N/A
Editar Borrar	6	tabindex	Una forma fácil de setear el tabindex de los input...	20030008	N/A
Editar Borrar	7	setInterval en componentes	estaba haciendo un componente, que tenía que despi...	20030013	N/A

Figura 7.6.

Contenidos de la tabla agenda de la base de datos pruebas.

Carpeta *includes*

Esta carpeta contiene los documentos de configuración (`config.php` y `funciones.php`) y la plantilla que se utiliza para imprimir las noticias de la agenda (`plantilla-imprimir.php`).

A continuación explicaremos los contenidos que han sido modificados del documento `funciones.php`, en el que hay cuatro funciones nuevas. Las dos primeras son utilizadas para crear los menús desplegables (que suelen ser utilizados para especificar días, meses y años). También contiene una función que proporciona formato a la fecha, recibiendo como parámetro una cadena de texto con

formato "aaaammdd" y retornando otra cadena de texto con el formato "día de mes, año". La última función que se ha añadido es la que genera el documento `agenda.xml`, cuyos datos son posteriormente accedidos por Flash.

Antes de entrar en detalle con cada una de las funciones que generan los menús desplegables, veamos cómo se crean estos utilizando código HTML:

```
<select name="nombre">
  <option value="valor"
selected>etiqueta</option>
  ...
  <option value="valor"
>etiqueta</option>
</select>
```

En esta declaración, los valores que pueden modificarse son el nombre del menú (`name`), el valor de cada una de ellas (`value`), así como su etiqueta (`etiqueta`), además de que podemos especificar (aunque no es obligatorio) que exista una opción seleccionada por defecto acompañándola con la palabra `selected`.

La función `makeNumList` genera una lista de números, recibiendo como parámetros los valores de la primera opción y de la última, el nombre del menú y la opción seleccionada. Primero creamos la etiqueta HTML `select`, junto con el nombre del menú (pasado por parámetro, línea 2).

Posteriormente, creamos un bucle `for`, en el que el valor de inicio es el parámetro `de`, y el valor final será el número resultante `de + cuantos` (línea 3).

Mientras estamos dentro del bucle se realizan dos comprobaciones:

- La primera consiste en que si el número resultante es menor que diez, se le agrega

un cero a la izquierda, ya que en el formato de fecha que estamos utilizando necesitamos dos cifras tanto para los días como para los meses (línea 4).

- La segunda comprobación se realiza comparando la igualdad entre el valor a escribir y el valor que ha de mostrarse como seleccionado (línea 6).

La utilidad de esta segunda comprobación es la de que cuando se ingrese un registro nuevo (en nuestro caso, una nueva noticia), el menú desplegable mostrará la fecha actual y la pasará por parámetro a la función, con lo que no habremos de tocar los menús desplegables para establecer la fecha de la noticia. Si, en cambio, estamos editando un registro, no perderemos su fecha, puesto que ésta ya aparecerá seleccionada.

En la línea 8 se cierra la etiqueta `select` y en la línea 9 se cierra la función. El código completo de la misma (que puede encontrar en el documento `material/cap7/proyecto/includes/funciones.php` del CD) es el siguiente:

```
1function makeNumList($de,$cuantos,$nombre,$seleccionado){
2  echo"\n\t<select name=\"\$nombre\">\n\t";
3  for ($n=$de; $n<($de+$cuantos);$n++){
4    $poner = ($n < 10)?($poner = "0".$n):($poner = $n);
5    $actual=($poner==$seleccionado)?(" selected"):"";
6    echo"\t\t<option value=\"\$poner\" $current>$poner</option>\n";
7  }
8  echo "\t</select>";
9} //fin makeNumList
```

Esta función nos es útil en el caso de que el menú desplegable contenga números que hayamos de seleccionar (como en el caso del día y del año), siendo los valores de la opción iguales a la etiqueta. Sin embargo, existe una

pequeña diferencia cuando hemos de mostrar el menú desplegable con los meses, en el que tomaremos los valores de las etiquetas de un *array* llamado *meses* externo a la función (línea 1).

Utilizar un *array* nos permite averiguar la cantidad de opciones que mostrará el menú (mediante el número de elementos del *array*), con lo que nuestra función sólo necesitará dos parámetros: nombre del menú y valor seleccionado (línea 3).

Puesto que el *array* es externo a la función, podemos acceder a sus datos convirtiéndolo en global (línea 4), y en caso de que no exista un valor seleccionado, establecemos como primer valor del menú la cadena de texto "00" (correspondiente a *Enero*), y como etiqueta, la cadena de texto "Seleccione" (línea 6).

El bucle *for* que utilizamos es similar al de la función anterior, pero tomando como número de opciones la longitud del *array* y como valor inicial cero (línea 7).

Posteriormente, comprobamos que el valor de la opción contenga dos dígitos (línea 8), ingresando un cero a la izquierda si sólo contiene uno, y revisamos también que exista un valor seleccionado. En la línea 10 se imprime la opción actual, utilizando el valor numérico del contador del bucle para obtener el nombre del mes almacenado en el *array* y utilizarlo como etiqueta. Finalmente, cerramos la etiqueta del menú (línea 12) y la función (línea 13). El código completo de la función (que puede encontrar en el documento *material/cap7/proyecto/includes/funciones.php* del CD) es el siguiente:

```
1  $meses = array ("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio",  
"Agosto","Septiembre","Octubre","Noviembre","Diciembre");  
2  //list box con los meses.  
3  function makeMesList($nombre,$seleccionado){  
4  global $meses;  
5  if(!isset($seleccionado)){ $poner = "seleccionado";}  
6  echo"\n\t<select name=\"$nombre\" >\n\t<option value=\"$00\" $poner>Seleccione</  
option>\n";  
7  for ($n=0; $n<sizeof($meses);$n++){  
8      $poner = ($n < 10)?($poner = "0".$n):($poner = $n);  
9      $current=($poner==$seleccionado)?(" selected"):"";  
10     echo"\t\t<option value=\"$poner\" $current>".$meses[$n]."</option>\n";  
11 }  
12 echo "\n\t</select>";  
13 }//fin makeMesList
```

La siguiente función que añadimos es la que da formato a la fecha, *formatFecha*, en la que el parámetro de entrada es una cadena de texto de ocho caracteres en la que las primeras cuatro letras corresponden al año, las dos siguientes al mes y las dos últimas al día. Ya que disponemos de un *array* con los nombres de los meses, volvemos a convertirlo en global para poder acceder a sus datos desde dentro

de la función (línea 2). Puesto que el *array* dispone de un índice numérico, no sólo hemos de extraer el texto correspondiente al mes (línea 3), sino también convertirlo a un número entero (línea 4), de igual forma que haremos con el día (líneas 5 y 6), lo que nos ayudará a eliminar los ceros cuando el valor sea menor que diez.

Cuando por fin hayamos obtenido estos datos, tan sólo nos resta retornar la fecha en el formato que queremos (línea 7) y en la que para el año, puesto que no necesitamos convertirlo a número entero, simplemente extraemos la cadena de texto. El código completo para esta función (que puede encontrar en el documento `material/cap7/proyecto/includes/funciones.php` del CD) es el siguiente:

```
1 function  formatFecha($fecha){
2     global  $meses;
3     $mes =substr($fecha, 4, 2);
4     settype($mes,"integer");
5     $dia = substr ($fecha, 6, 2);
6     settype($dia,"integer");
7     return ($dia . " de ". $meses[$mes]. ", " . substr ($fecha, 0, 4));
8 } // end func
```

La última función que añadiremos a `funciones.php` es la que escribe el documento XML para la agenda (`agenda.xml`). Los pasos a seguir para generar dicho documento son los siguientes:

- Seleccionar sólo las columnas `id`, `cabe-cera`, `fecha` y `foto`.
- Ordenar estos datos en orden descendente, primero por `fecha` y luego por `id`, de modo que los registros más recientes se mostrarán primero.

Tenga en cuenta que pueden existir varias noticias con la misma fecha, las cuales han de pertenecer al mismo nodo del árbol XML, y también puede darse el caso de que existan registros sin imagen asociada a ellos.

Lo primero que hacemos en nuestra función es establecer la dirección del documento XML que contendrá la información, en este caso, en el directorio padre de aquél en el que está la

carpeta `includes` (`proyecto`, línea 2) para a continuación especificar la consulta SQL a la base de datos (línea 4).



Nota: Observe que no se ha establecido conexión con la base de datos, ya que esta función será invocada dentro de las páginas que modifiquen registros y que ya habrán establecido previamente la conexión con MySQL. Hemos de tomar la prevención de realizar la llamada a esta función antes de haber cerrado la conexión en estas páginas.

En caso de que existan resultados tras la consulta (línea 5), asignamos el valor de la etiqueta inicial del nodo raíz del árbol XML a la variable `$salida` (línea 6).

Antes de continuar con la recogida de datos del resultado de la consulta, revisemos la

estructura que deseamos almacenar en el documento XML (figura 7.7).

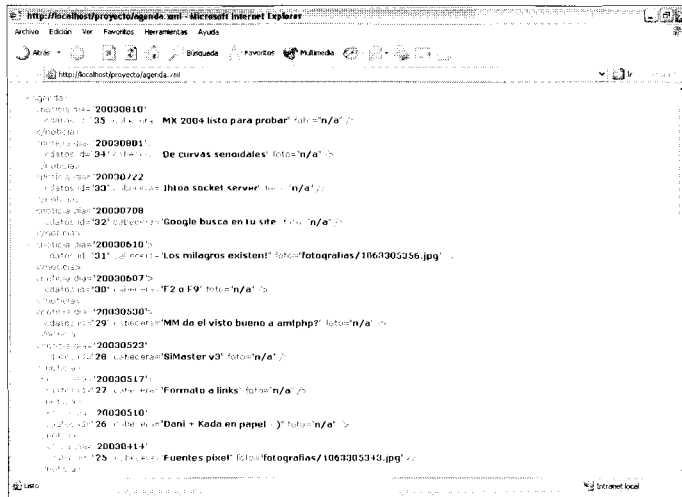


Figura 7.7.
Contenidos del documento *agenda.xml*.

```
<agenda>
<noticia dia='{fecha}'>
  <datos id='{id}' cabecera='{cabecera}' foto='{foto}' />
</noticia>
<noticia dia='{fecha}'>
  <datos id='{id}' cabecera='{cabecera}' foto='{foto}' />
  <datos id='{id}' cabecera='{cabecera}' foto='{foto}' />
</noticia>
</agenda>
```

Por cada día en el que haya una noticia se crea un nodo de nombre *noticia*, con la fecha como único atributo. Se imprimen los datos de la noticia en el interior de este nodo, cerrándolo cada vez que la fecha de la siguiente noticia que se lea varíe con respecto a la del nodo que esté abierto en ese instante.

Para detectar este cambio en la fecha, asignaremos el valor de la fecha actual en el bucle a una variable y la compararemos con el valor que tenía en la anterior iteración. Si se ha producido un cambio, cerramos el nodo y abrimos uno nuevo, motivo por el cual inicializamos la variable *\$cfecha* con el valor 0 (línea 8) antes de entrar en el bucle, para así

poder detectar un cambio en la fecha desde la primera noticia a la que se acceda.

Una vez en el interior del bucle, asignamos los valores de la fila actual a las variables *\$id*, *\$cabecera*, *\$fecha* y *\$foto* (línea 9) y comprobamos el valor contenido en la variable *\$cfecha*. Si el valor que contiene es 0, significa que estamos en el primer resultado, con lo que sólo es necesario abrir una nueva etiqueta para el nodo *noticia* (junto con el valor de la variable *\$fecha* como contenido de su atributo *dia*). En caso contrario, se cierra el nodo *noticia* anterior y se abre uno nuevo (líneas 13-19) y asignamos el valor de la fecha que actualmente existe en el bucle (*\$fecha*) a la variable *\$cfecha*.

Antes de imprimir los datos en el nodo, hemos de comprobar si el registro dispone o no de una imagen asociada, lo que podemos averiguar si el valor almacenado en `$foto` es diferente a "N/A" o, simplemente, el registro está vacío. En caso de que exista una imagen, adjuntamos en la variable `$fotoUrl` el nombre de ésta junto con la ruta a la carpeta en la que están contenidas (fotografías). Si no existe imagen alguna, establecemos como contenido de `$fotoUrl` la cadena de texto "n/a", que Flash interpretará posteriormente (línea 26).

En la línea 27 concatenamos los datos contenidos en la variable `$salida` con los datos del registro actual, cerramos el bucle `while`

en la línea 28, y en la línea 30 imprimimos las etiquetas que cierran tanto el nodo actual (*noticia*) como el nodo principal (*agenda*).

Para finalizar, abrimos el fichero XML para escritura (línea 32), se escriben los datos que contiene la variable `$salida` (línea 34) y se cierra el fichero (línea 36).

En caso de no obtener resultados en la consulta, eliminamos el fichero físicamente (líneas 37-44) y posteriormente cerramos la función. El código completo para la misma (que puede encontrar en el documento `material/cap7/proyecto/includes/funciones.php` del CD) es el siguiente:

```
1function actualizarXmlAgenda(){
2  //nombre del xml
3  $agendaxml = "../agenda.xml";
4  $res = mysql_query("SELECT id,cabecera,fecha,foto FROM agenda ORDER BY
fecha DESC,id DESC")or die(mysql_error());
5  if( mysql_num_rows ($res) > 0 ){
6      $salida = "<agenda>\n";
7      //organidor por fecha
8      $cfecha = 0;
9      while(list($id,$cabecera,$fecha,$foto) = mysql_fetch_array($res)){
10         //revisamos si hay cambio de fecha.
11         if($fecha != $cfecha){
12             //si es la primer fecha.
13             if($cfecha == 0){
14                 //es la primer etiqueta.
15                 $salida .= "\t<noticia dia='$fecha'>\n";
16             }else{
17                 //no es la primer fecha, cerramos la etiqueta anterior
y ponemos la nueva.
18                 $salida .= "\t</noticia>\n\t<noticia dia='$fecha'>\n";
19             }
20             //actualizamos cfecha.
21             $cfecha = $fecha;
22         }//fin if de fecha.
23
24         //datos normales.
25         //revisamos si tiene o no imagen.
26         $fotoUrl = ($foto == "" || $foto == N/A)?("n/a"):(fotografias/
$foto);
27         $salida .= "\t\t<datos id='$id'
cabecera='".utf8_encode($cabecera)."' foto='$fotoUrl' />\n";
```

```

28         }//fin del while
29     //fin del xml
30     $salida .= "\t</noticia>\n</agenda>";
31     //abrimos el fichero.
32     $fp = fopen($agendaxml,"w");
33     //escribimos el contenido de $salida en el.
34     fwrite($fp,$salida);
35     //cerramos el fichero
36     fclose($fp);
37 }else{
38     //no hay registros borramos la agenda
39     //revisamos si el archivo existe
40     if( is_file($agendaxml)){
41         //si existe lo borramos
42         unlink ($agendaxml);
43     }
44 } //fin if/else de resultados
45 } // end fun.
46?>

```

Carpeta *admin*

En esta carpeta se encuentran los ficheros que dan forma física al Panel de Administración, que son los documentos que se utilizarán para modificar la información en la base de datos y los que servirán para crear accesos restringidos mediante un nombre de usuario y su correspondiente contraseña.

Documentos de creación de zonas de acceso restringido

Los documentos de creación de zonas de acceso restringido son los mismos que se han explicado al inicio de este capítulo, excepto `interface.php` (el que contiene el formulario de ingreso al sistema), al cual se le ha aplicado una hoja de estilos (el documento `estilos.css` del directorio `proyecto`) para darle una apariencia acorde al aspecto que se ha utilizado a la hora de diseñar la agenda.

Documento principal

El documento principal de esta carpeta es `index.php`, en el que se muestran todos los registros almacenados en la agenda, cada uno de los cuales con enlaces a las páginas de ver, editar y borrar (figura 7.8).

En las líneas 2 y 3 se incluyen los documentos de configuración y en la línea 4 se incluye `secure.php`, el fichero que se encargará de comprobar que el usuario haya ingresado en el sistema antes de ver el resto de la página.

La consulta SQL se realiza seleccionando únicamente las columnas *cabecera* e *id* (línea 7). En este documento, como en los demás de este directorio, se incluye el mismo archivo de estilos en cascada (línea 14) para darle una apariencia más vistosa y organizada a los documentos de gestión de la agenda.

Cabecera			Acción		
35	Mir 2004 foto para probar	ver	editar	borrar	
34	De curules senadores	ver	editar	borrar	
33	Mitos pocket server	ver	editar	borrar	
32	Google busca en tu site	ver	editar	borrar	
31	Los riesgos existen	ver	editar	borrar	
30	P2 o P3	ver	editar	borrar	
29	Mir 2004 el video bueno a entip?	ver	editar	borrar	
28	Sistemas G3	ver	editar	borrar	
27	Farmacia Link	ver	editar	borrar	
26	Dati + Maca en papel +	ver	editar	borrar	
25	Fuentes oval	ver	editar	borrar	
24	Exoticos en Bash M2 (es)	ver	editar	borrar	
23	LocalConnection + Mac + J2-entrip + pan	ver	editar	borrar	
22	FlashMovieing + mto	ver	editar	borrar	
21	Rin de force y paralelos	ver	editar	borrar	
20	Real gracias a todos	ver	editar	borrar	
19	random_3_1_1	ver	editar	borrar	
18	La cabecera final	ver	editar	borrar	
17	display a todos los niveles	ver	editar	borrar	
16	What is it?	ver	editar	borrar	
15	Variables de solo lectura	ver	editar	borrar	
14	Amigos	ver	editar	borrar	
13	De círculos y triángulos	ver	editar	borrar	
12	mi FlashFormacion	ver	editar	borrar	
11	Exoticos en Bash	ver	editar	borrar	
10	(code) sacando el código (code)	ver	editar	borrar	
9	subinterval vs interval range	ver	editar	borrar	

Figura 7.8.
*Documento principal
index.php.*

Si la consulta retorna resultados, los imprimimos (líneas 24-37). En caso contrario, informamos al usuario (líneas 38-45) de tal circunstancia. Observe que en el enlace que apunta a la página de eliminar registros

(agenda_eliminar.php) se incluye también la cabecera del registro (línea 34), para que en vez de mostrar el mensaje de alerta en base al identificador único `id` se utilice la cabecera de la noticia (figura 7.9).

Eliminar Registro

¿Eliminar registro: Amigos?

Figura 7.9.
*Mensaje de alerta utilizando
la cabecera de la noticia.*

El código completo de este documento (que puede encontrar en la carpeta material/cap7/proyecto/admin del CD) es el siguiente:

```
1 <?
2 include("../includes/config.php");
3 include("../includes/funciones.php");
4 include("secure.php");
5
6 $cnx = conectar();
7 $res = mysql_query("SELECT id,cabecera FROM agenda ORDER BY fecha DESC,id
DESC") or die (mysql_error());
8 ?>
9
10<html>
11<head>
12<title>agenda</title>
13<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
14<link href="../estilos.css" rel="stylesheet" type="text/css">
15</head>
16
17<body>
18<table width="600" border="0" align="center" cellpadding="0" cellspacing="0">
19  <tr class="cabeceraBold">
20    <td>id</td>
21    <td width="300" height="30" class="cabeceraBold">Cabecera</td>
22    <td colspan="3" align="center" class="cabeceraBold">acci&oacute;n</td>
23  </tr>
24  <?
25    if (mysql_num_rows($res) > 0) {
26      //si hay resultados.
27      while(list($id,$cabecera) = mysql_fetch_array($res)){
28        ?>
29  <tr>
30    <td class="texto"><? echo $id;?></td>
31    <td class="texto"><? echo $cabecera;?></td>
32    <td align="center" class="pie"><a href="agenda_ver.php?id=<? echo
$id;?>">ver</a></td>
33    <td align="center" class="pie"><a href="agenda_editar.php?id=<? echo
$id;?>">editar</a></td>
34    <td align="center" class="pie"><a href="agenda_borrar.php?id=<? echo
$id;?>&cabecera=<?echo $cabecera;?>">borrar</a></td>
35  </tr>
36    <?
37  }//fin del while
38  }else{
39    //no hay resultados
40    ?>
41  <tr>
42    <td colspan="5" align="center">No hay datos</td>
43  </tr>
44    <?
45  }//fin del if/else de resultados
46  ?>
47  <tr>
```

```

48 <td height="30" colspan="4" class="pie"><a
href="agenda_agregar.php">Agregar registro</a></td>
49 <td height="30" align="right" class="pie"> <a href="salir.php">salir</a></
td>
50 </tr>
51</table>
52</body>
53</html>

```

Agregar un registro

El documento que utilizaremos para añadir registros a la agenda es `agenda_agregar.php`, en el que hay un formulario que se utiliza para introducir los datos (figura 7.10). Cuando éste es enviado, se comprueba si junto con las variables se ha enviado algún documento de imagen; en caso de ser así, se le cambia el nombre por uno único y se almacena en la variable `$foto` (líneas 9-30). Si existe algún error en la carga de la imagen, se asigna a dicha variable la cadena de texto "N/A" (líneas 28-30).

Tras realizar el ingreso de los datos (línea 39) y posterior cierre de la conexión (línea 43), se

configuran tres variables: `$titulo`, `$mensaje` y `$link` (líneas 45-47). El motivo de declarar estas tres variables es el de que el mensaje de éxito en la operación no se imprime desde esta página, sino que se incluye una nueva (`mensajes.php`, línea 48) que espera el valor de las tres variables recién declaradas para mostrar el mensaje de éxito.

Este nuevo documento que se incluye, `mensajes.php`, es una página con una tabla que contiene los formatos de la hoja de estilo y que imprime el resultado del ingreso o modificación del registro, lo que es más apropiado que escribir el código que genera la página HTML directamente en PHP.

The screenshot shows a web browser window titled 'agenda_agregar - Microsoft Internet Explorer'. The address bar shows 'http://localhost/proyecto/admin/agenda_agregar.php'. The form contains the following elements:

- A header field labeled 'Cabecera:'.
- A text area labeled 'Texto:'.
- A date field labeled 'Fecha:' with a dropdown menu showing '19' and a month/year selector showing 'Septiembre' and '2007'.
- A file upload field labeled 'Foto:' with a 'Eliminar' button next to it.
- An 'Enviar' button at the bottom right.
- A 'regresar' link at the bottom left.

Figura 7.10.
Formulario para agregar una noticia.

En la parte del documento que contiene código HTML puede observar que el formulario es de tipo multiparte (línea 62), ya que es necesario proporcionar la posibilidad de subir una imagen al servidor (línea 99).

Para generar los menús para las fechas se utilizan las funciones declaradas en el documento de configuración `funciones.php`. El menú que corresponde al día de la noticia (línea 83) realiza una llamada a la función `makeNumList` para que genere opciones de 1 a 31, con nombre `dd` y con un valor seleccionado equivalente al del día actual (`date("d")` retorna el número del día del mes utilizando dos dígitos).

A la hora de crear el menú para seleccionar los meses, asignamos primeramente el valor del mes actual (restándole una unidad) y posteriormente se comprueba si necesita o no un cero delante para completar los dos dígitos

(línea 87). Realizamos una llamada a la función `makeMesList` con el valor del mes recién realizado, junto con el nombre que deseamos para el menú, `mm` (línea 88).

En el caso del año se utiliza la misma función que para el día, con la diferencia de que tomamos como número de partida dos años anteriores al año en curso (por si se da el caso de que existan registros con fechas antiguas), 6 como la cantidad de años a mostrar, `aa` como nombre de menú, y el año actual como parámetro seleccionado.

A pie de página existe un enlace que apunta al documento principal del administrador (`index.php`) por si no desea ingresar un nuevo registro.

El código completo del documento `agenda_agregar.php` (que puede encontrar en la carpeta `material/cap7/proyecto/admin del CD`) es el siguiente:

```
1<?
2include("../includes/config.php");
3include("../includes/funciones.php");
4include("secure.php");
5
6if(isset($_POST['submit'])) {
7    $error = false;
8    // si hay imagen.
9    if (is_uploaded_file($_FILES['imagen']['tmp_name'])) {
10    //revisamos que sea jpg
11    if ($_FILES['imagen']['type'] == "image/jpeg" || $_FILES['imagen']['type']
12    == "image/pjpeg") {
13    //nombre de la imagen
14    $foto = time()."jpg";
15    //movemos la imagen.
16    move_uploaded_file($_FILES['imagen']['tmp_name'], "../fotografias/".$foto);
17    } else {
18    $error = true;
19    $errmsg = "Formato no válido para archivo de imagen";
20    }
21    } else {
22    //imagen no se pudo subir o no seleccionaron.
23    $error=true;
24    $errmsg = "Error al cargar imagen: " . $_FILES['imagen']['name'];
25    }
26    }
27    }
28    }
29    }
30    }
31    }
32    }
33    }
34    }
35    }
36    }
37    }
38    }
39    }
40    }
41    }
42    }
43    }
44    }
45    }
46    }
47    }
48    }
49    }
50    }
51    }
52    }
53    }
54    }
55    }
56    }
57    }
58    }
59    }
60    }
61    }
62    }
63    }
64    }
65    }
66    }
67    }
68    }
69    }
70    }
71    }
72    }
73    }
74    }
75    }
76    }
77    }
78    }
79    }
80    }
81    }
82    }
83    }
84    }
85    }
86    }
87    }
88    }
89    }
90    }
91    }
92    }
93    }
94    }
95    }
96    }
97    }
98    }
99    }
100   }
```



```

79 </tr>
80 <tr>
81 <td><span class="textoBold">Fecha:</span><br>
82 <?
83 makeNumList(1,31,"dd",date("d"));
84 ?>
85 -
86 <?
87 $m = (( $m =(date("n") -1))<10)?("0".$m):($m);
88 makeMesList("mm",$m);
89 ?>
90 -
91 <?
92 makeNumList(date("Y") -2,6,"aa",date("Y"));
93 ?>
94 </td>
95 </tr>
96 <tr>
97 <td><span class="textoBold">Foto:</span> <input type="hidden"
name="MAX_FILE_SIZE" value="100000">
98 <br>
99 <input name="imagen" type="file" id="imagen"></td>
100 </tr>
101 <tr>
102 <td>&nbsp;</td>
103 </tr>
104 <tr>
105 <td align="right"><input name="submit" type="submit" id="submit"
value="Enviar"></td>
106 </tr>
107 <tr>
108 <td height="30" class="pie"><a href="index.php">regresar</a></td>
109 </tr>
110 </table>
111 </form>
112 </body>
113 </html>

```

Ver un registro

Aparte de la inclusión del documento `secure.php` (línea 4), el enlace de regreso al documento principal (línea 60), este documento (figura 7.11) es idéntico a la plantilla de

impresión que explicaremos posteriormente cuando expliquemos la realización en Flash de la interfaz de la agenda.

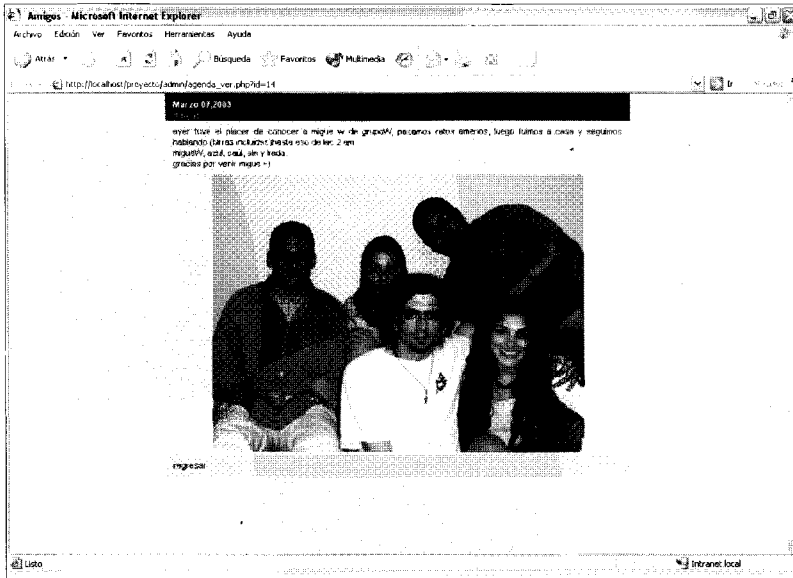


Figura 7.11.
Visualización de una noticia.

El código completo para este documento (que puede encontrar en la carpeta material/

cap7/proyecto/admin del CD) es el siguiente:

```
1<?
2include("../includes/config.php");
3include("../includes/funciones.php");
4include("secure.php");
5
6if(empty($_GET['id'])){
7    //no hay id
8    header ("Location: index.php");
9    exit;
10}else{
11    //nos conectamos a la bd.
12    $cnx = conectar();
13
14    $res = mysql_query("SELECT cabecera,texto,fecha,foto FROM agenda WHERE id =
15    \"$_GET['id']\" or die(\"&output=error&msg=\".mysql_error());
16    if( mysql_num_rows($res) > 0 ){
17        //si hay datos.
18        list($cabecera,$texto,$fecha,$foto) = mysql_fetch_array($res);
19        //formateamos la fecha.
20        $fecha = formatFecha($fecha);
21        //cambiamos los \n por <br> para html
22        $texto = nl2br($texto);
23    }else{
24        //no hay datos.
25        echo "error al buscar el registro.";
26        exit;
27    }
28    //cerramos la conexion con mysql.
```

```

29  mysql_close($cnx);
30}
31?>
32<html>
33<head>
34<title>
35<? echo $cabecera;?>
36</title>
37
38<link href="../../../estilos.css" rel="stylesheet" type="text/css">
39</head>
40<body>
41<table width="600" height="350" border="0" align="center" cellpadding="10"
cellspacing="0">
42 <tr height="30">
43 <td valign="top" class="cabecera"> <span class="cabeceraBold"><? echo
$fecha;?></span><br>
44 <? echo $cabecera;?></td>
45 </tr>
46 <tr>
47 <td valign="top" class="texto"><?echo $texto?><br></td>
48 </tr>
49 <?
50 if($foto != "N/A" && !empty($foto)){
51     //hay foto.
52     ?>
53 <tr>
54 <td align="center"> "> </td>
55 </tr>
56 <?
57 }//fin if de foto
58 ?>
59 <tr height="30">
60 <td valign="top" class="pie"><a href="index.php">regresar</a></td>
61 </tr>
62</table>
63</body>
64</html>

```

Editar un registro

El documento que se utiliza para editar registros (`agenda_editar.php`) crea una variable de nombre `imgpath` que almacena la ruta hacia la carpeta en la que se guardan las imágenes (línea 7). Después se comprueba que exista la variable `submit` y que haya sido enviada mediante el método `POST` (línea 9), para continuar con la actualización del registro afectado por la edición (líneas 12-21).

Posteriormente se actualiza el documento `agenda.xml` (línea 23) y se asignan las variables para incluir el documento que muestra el mensaje de éxito de la operación (líneas 27-30).

En la línea 34 se comprueba la existencia de la variable `id` y el que haya sido enviada mediante el método `GET`, realizando a continuación la consulta a la base de datos.

En la cabecera de la parte del documento que contiene código HTML se han declarado dos funciones de JavaScript: `openImageManager` (líneas 48-53), que sirve para abrir la galería de imágenes y `eliminarImagen` (líneas 54-58), la cual explicaremos más adelante en este mismo apartado.

Entre las líneas 72 y 123 encontramos la declaración de un formulario que muestra la información actualmente contenida en el registro que estamos editando en campos de texto y menús de lista en el caso de la fecha (líneas 93-102), siendo los valores seleccionados los provenientes de la base de datos.

Puede que cuando se edite una noticia almacenada en la agenda (figura 7.12) deseemos

eliminar la imagen que la acompaña (si dispone de ella) o bien añadir una nueva que complemente la noticia (si no dispone de ella). Para esta labor, no sería apropiado que cada vez que se desee modificar una noticia haya que subir de nuevo la imagen ni dejar disponible un campo de texto para introducir manualmente el nombre de la foto, ya que pueden producirse errores. Por ello hemos añadido un campo de texto de sólo lectura (línea 108) en el que se mostrará el nombre de la imagen, pero el que realmente tendremos en cuenta es un campo oculto (línea 109) que, por esta misma razón, no puede ser modificado (debido a que algunos navegadores no admiten la propiedad de sólo lectura para un campo de texto).



Figura 7.12.
Formulario de edición de una noticia.

Anteriormente mostramos la imagen actual (línea 107) con un identificador único (`id=imageDisplay`) con el propósito de identificar los distintos campos (tanto la imagen como los campos de texto que contienen el nombre de ésta), para que cuando seleccionemos una imagen desde la galería podamos visualizar los cambios directamente en la página una vez que los realicemos.

Por otra parte, al presionar el enlace que ejecuta la función JavaScript `eliminarImagen` (línea 110), la imagen no será visible, ya que cambiaremos la propiedad `source` (que

contiene la ruta hasta la imagen) de ésta, a la vez que asignamos la cadena de texto "N/A" al campo de texto de sólo lectura (cuyo nombre es `foto2`, línea 108) y al campo oculto (cuyo nombre es `foto`, línea 109).

En las líneas 110 y 111 hemos añadido un enlace que llama a la otra función *JavaScript*, `openImgManager`, que abre en una ventana flotante la galería de imágenes.

El código completo para el documento `agenda_editar.php` (que puede encontrar en la carpeta `material/cap7/proyecto/admin` del CD) es el siguiente:

```
1  <?
2  include("../includes/config.php");
3  include("../includes/funciones.php");
4  include("secure.php");
5
6  //path de las imagenes
7  $imgpath = "../fotografias/";
8
9  if(isset($_POST['submit'])){
10     //actualizamos el registro.
11
12     //cambiamos los enter por nuevas lineas
13     $noticia = str_replace("\r","",$_POST['texto']);
14     //fecha
15     $fecha = $_POST['aa'].$_POST['mm'].$_POST['dd'];
16     $sql = "UPDATE agenda SET ";
17     $sql .= "cabecera='".$_POST['cabecera']."' ,texto='$noticia',
18 fecha='$fecha',foto='".$_POST['foto']."' ";
19     $sql .= "WHERE id= ".$_POST['id'];
20     //nos conectamos a la bd.
21     $cnx = conectar();
22     $res = mysql_query($sql) or die (mysql_error());
23     //actualizamos el xml de agenda.
24     actualizarXmlAgenda();
25     //cerramos la conexión.
26     mysql_close($cnx);
27     //mensaje de éxito.
28     $titulo = "Registro Actualizado";
29     $mensaje = "El registro ha sido Actualizado";
30     $link = "<a href='index.php'>regresar</a>";
31     include("mensajes.php");
32     exit;
33 }
34
35 if(empty($_GET['id'])){
36     header("Location: index.php");
```

```

36     }
37     $cnx = conectar();
38     $res = mysql_query ("SELECT * FROM agenda WHERE id = ".$_GET['id']) or die
(mysql_error());
39
40     ?>
41
42     <html>
43     <head>
44         <title>agenda_editar</title>
45         <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
46         <SCRIPT LANGUAGE="JavaScript">
47         <!--
48         var galeria = null;
49         function openImgManager() {
50             if(galeria && !galeria.closed) galeria.close();
51             var left = screen.width - 465;
52             galeria =
window.open('imgMgr.php','galeria','left='+left+',top=100,status=yes,scrollbars=yes,width=450,height=500');
53         }
54         function eliminarImagen(){
55             document.form1.foto.value = "N/A";
56             document.form1.foto2.value = "N/A";
57             document.form1['imgDisplay'].src = "N/A";
58         }
59         //-->
60     </SCRIPT>
61     <link href="../estilos.css" rel="stylesheet" type="text/css">
62     </head>
63
64     <?
65         if(mysql_num_rows($res)> 0) {
66             //si hay datos
67             list($id,$cabecera,$texto,$fecha,$foto) = mysql_fetch_array($res);
68
69             ?>
70
71             <body>
72                 <form action="<? echo $SERVER['PHP_SELF']?>" method="post" name="form1">
73                     <table width="500" border="0" align="center" cellpadding="0"
cellspacing="0">
74                         <tr>
75                             <td height="30" class="cabeceraBold"><input type="hidden" name="id"
value="<? echo $id;?>">
76                             Editar noticia en la agenda.      </td>
77                         </tr>
78                         <tr>
79                             <td>&nbsp;   </td>
80                         </tr>
81                         <tr>
82                             <td><span class="textoBold">Cabecera:</span><br>
83                             <input name="cabecera" type="text" id="cabecera" size="40" value="<? echo
$cabecera;?>">
84                             </td>
85                         </tr>
86                         <tr>
87                             <td><span class="textoBold">Texto:</span><br>

```

```

88      <textarea name="texto" cols="40" rows="6" id="texto"><? echo $texto;?></
textarea>
89      </td>
90      </tr>
91      <tr>
92          <td><span class="textoBold">Fecha:</span><br>
93              <?
94                  makeNumList(1,31,"dd",substr($fecha,6,2));
95              ?>
96          -
97          <?
98              makeMesList("mm",substr($fecha,4,2));?>
99          -
100         <?
101             makeNumList(date("Y") -2,6,"aa",substr($fecha,0,4));
102         ?>
103     </td>
104 </tr>
105 <tr>
106     <td><span class="textoBold">Foto</span><br>
107         <div align="center"><br>
109         <input name="foto2" readonly type="text" id="foto2" value="<? echo
110 $foto;?>">
111         <input name="foto" type="hidden" id="foto" value="<? echo $foto;?>">
112         <br><a href="javascript:;" onClick="eliminarImagen()">eliminar imagen</
113 a>&nbsp; &nbsp;<a href="javascript:;" onClick="openImgManager()">escoger
114 imagen de
115 galer&iacute;a </a> </div>
116 </tr>
117 <td>&nbsp;</td>
118 </tr>
119 <tr>
120     <td align="right"><input name="submit" type="submit" id="submit"
121 value="Enviar"></td>
122 </tr>
123 <tr>
124     <td height="30" class="pie"><a href="index.php">regresar</a></td>
125 </tr>
126 </table>
127 </form>
128 <?
129 }else{
130     //no hay datos
131     echo "No hay registros que coincidan con el identificador";
132 }
133 mysql_close($cnx)
134 ?>
135 </body>
136 </html>

```

Galería de imágenes

En la galería de imágenes (`imgMngr.php`), el cambio más significativo con respecto al administrador que realizamos al comienzo de este capítulo es el de que podemos comunicarnos con la página de edición de registros mediante la función de JavaScript `fillInfo`, que recibe como parámetro el nombre de la imagen y cambia tanto la imagen como las propiedades de los campos del formulario en el documento `agenda_editar.php`.

Para conseguir establecer esta comunicación entre los dos documentos, a la hora de mostrar las imágenes en la galería (líneas 67-85), se incluye la posibilidad de realizar una llamada a

la función `fillInfo` si pulsamos sobre la imagen (`javascript:fillInfo('$file')`, línea 75).

Este documento también permite incorporar una nueva imagen al servidor en caso de que ninguna de las disponibles nos sirva para complementar las noticias existentes en la agenda, para lo que cuando PHP sube la nueva imagen al servidor (líneas 9-34) actualiza el contenido de la galería cuando la página se carga de nuevo.

El código completo para el documento `imgMngr.php` (que puede encontrar en la carpeta `material/cap7/proyecto/admin` del CD) es el siguiente:

```
1  <?php
2  include ("../includes/config.php");
3  include ("../includes/funciones.php");
4  //carpeta de imagenes
5  $imgpath = "../fotos/grafias/";
6
7  //sube archivos
8
9  if( isset($_POST['submit'])) {
10     if (is_uploaded_file($_FILES['imagen']['tmp_name'])) {
11         //echo $_FILES['imagen']['type'];
12         //revisamos que sea jpg
13         if ($_FILES['imagen']['type'] == "image/jpeg"
14             $_FILES['imagen']['type'] == "image/pjpeg") {
15             $newName = time().".jpg";
16             if(!copy($_FILES['imagen']['tmp_name'], $imgpath.$newName)) {
17                 // no se puedo subir :S
18                 $error=true;
19                 $errmsg = "Error al cargar imagen: " .
20                 $_FILES['imagen']['name'];
21             }
22             //echo "$newName subido con exito";
23             //seguimos con el insert =)
24         } else {
25             $error = true;
26             $errmsg = "Formato no válido para archivo de imagen";
27         }
28     } else {
29         $error=true;
30         if($_FILES['imagen']['name'] == "") {
31             $errmsg = "No seleccionó imagen";
32         } else {
33             // ...
34         }
35     }
36 }
```

```

31         $errmsg = "al cargar imagen: " . $_FILES['imagen']['name'];
32     }
33 } //fin con o sin imagen
34 }
35 ?>
36
37 <html>
38 <head>
39     <title>galería de imágenes</title>
40     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
41     <SCRIPT LANGUAGE="JavaScript">
42     <!--
43     function fillInfo(imgPath){
44         window.opener.document.form1.foto.value = imgPath;
45         window.opener.document.form1.foto2.value = imgPath;
46         window.opener.document.form1['imgDisplay'].src = '<?echo $imgpath;?>' +
imgPath;
47     }
48     //-->
49     </SCRIPT>
50     <link href="../estilos.css" rel="stylesheet" type="text/css">
51 </head>
52 <?
53     //*****
54     //lee el directorio y despliega todos los archivos.
55     //
56
57     $handle=opendir($imgpath);
58     $count=0;
59     ?>
60
61     <table width="400" border="0" cellpadding="0" cellspacing="0"
align="center">
62         <tr>
63             <td height="30" colspan="3" align="center" class="cabeceraBold">Para
seleccionar una imagen haga click sobre ella</td>
64         </tr>
65         <tr>
66             <td>
67                 <!-- tabla para las imagenes -->
68                 <table width="400" border="0" cellpadding="0" cellspacing="0"
align="center">
69                     <tr>
70                         <?
71                         while ($file = readdir($handle)) {
72                             if ($file != "." && $file != "..") {
73                                 $fichero = $imgpath.$file;
74                                 $fileData = GetImageSize($fichero);
75                                 echo "\n<td align=\"center\" width =\"33%\" ><a
href=\"javascript:;\"><img src=\"\"$fichero\"/ width=100 height =100 border=0
onClick=\"javascript:fillInfo('$file')\"></a><br>$file
<br>(\".$fileData[0].\"x\". $fileData[1].)</td>";
76                                 $count++;
77                                 if($count==3){
78                                     $count = 0;
79                                     echo "</tr><tr>";
80                                 }

```

```

81     }
82 }
83     closedir($handle);
84 ?>
85     </table>
86     </td>
87 </tr>
88 <tr>
89     <td height="30" class="pie"><a href="javascript:window.close()"
>Cerrar ventana</a></td>
90 </tr>
91 <tr>
92     <td>
93         <form action="<? echo $PHP_SELF;?>" method="post"
enctype="multipart/form-data" name="form1" >
94         <table width="400" border="0" cellspacing="0" cellpadding="0"
align="center">
95             <tr>
96                 <td align="center" >
97                     <div class="textoBold">
98                         <?
99                             if($error){
100                                 echo "Error: ".$errorMsg;
101                             }
102                         ?>
103                     </div>
104                     <br>si no esta la imagen que busca puede grabarla Al
servidor<br>
105                         <input type="hidden" name="MAX_FILE_SIZE" value="100000">
106                         Subir imagen (solo .jpg*):<br> <input name="imagen"
type="file" id="imagen">
107                     </td>
108                 </tr>
109                 <tr>
110                     <td align="center" ><input name="submit" type="submit"
value="Subir"></td>
111                 </tr>
112             </table>
113         </form>
114     </td>
115 </tr>
116 </tr>
117 <tr>
118     <td height="30" class="pie"><a href="javascript:window.close()" >Cerrar
ventana</a></td>
119 </tr>
120 </table>
121 </body>
122 </html>

```

Realización en Flash de la interfaz para la agenda

Una vez que hemos construido toda la zona de administración de los contenidos, vamos a utilizar Flash para acceder a los mismos. Entre los documentos que habrá copiado con anterioridad en la carpeta *proyecto* de la raíz de su servidor encontrará uno de nombre *agenda.html*. Ábralo en su navegador tecleando `http://localhost/proyecto/agenda.html` para que pueda observar la forma en que la interfaz accede a la información contenida en la base de datos (figura 7.13).

que la complemente). En cambio, la zona inferior consta de un formulario de contacto en el que los usuarios pueden enviar sus opiniones a la dirección de correo electrónico que el administrador de la agenda elija.

El menú de noticias extrae la información del documento *agenda.xml* que se genera y actualiza cada vez que se modifican los datos de la tabla *agenda*. Flash accede a dicho documento y muestra la cabecera de cada una de las noticias por orden de menor a mayor antigüedad. Cuando pulsamos sobre una de ellas, utilizamos el documento

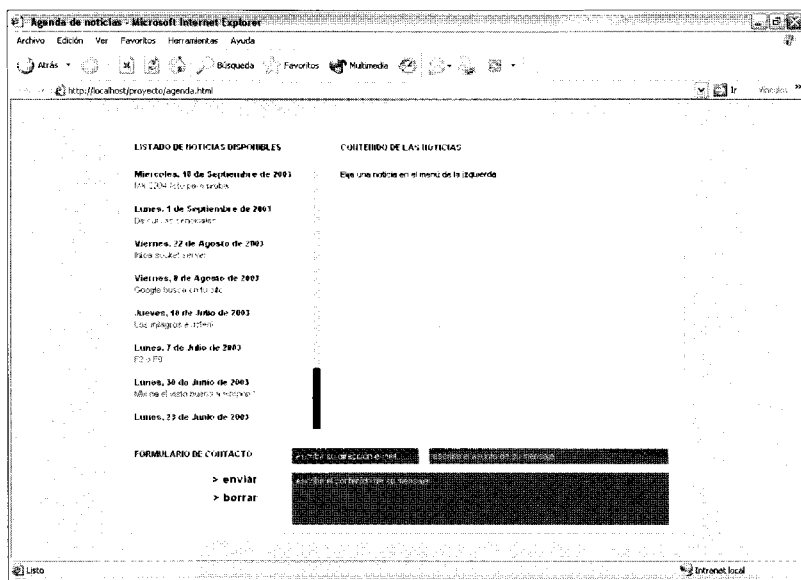


Figura 7.13.

Aspecto de la interfaz que maneja los datos de la agenda.

La interfaz se divide en dos zonas. La superior muestra en su parte izquierda los datos recogidos del documento *agenda.xml* y queda a la espera de la pulsación de alguna de las distintas noticias para mostrar su contenido en la parte derecha (incluyendo la foto en caso de que la noticia disponga de una

contenidos.php para enviarle el identificador único (*id*) de la noticia, esperando que devuelva el texto completo para la misma, que se mostrará en la parte derecha de la agenda, con la distinción de si existe foto o no para esa noticia. Si existe, el texto de la noticia se muestra en la columna central de la agenda, dejan-

do el lado derecho para la foto (figura 7.14). Si no existe fotografía para esa noticia, el contenido textual de la misma ocupa completamente la parte derecha (figura 7.15).

Una vez que se ha accedido tanto a la foto (si existe) como a los contenidos de la noticia, hemos incluido la posibilidad de imprimir cada una de ellas. Con este propósito incluimos unas líneas más abajo del final del texto de la noticia un enlace con el texto "imprimir noticia", que al ser pulsado abre una

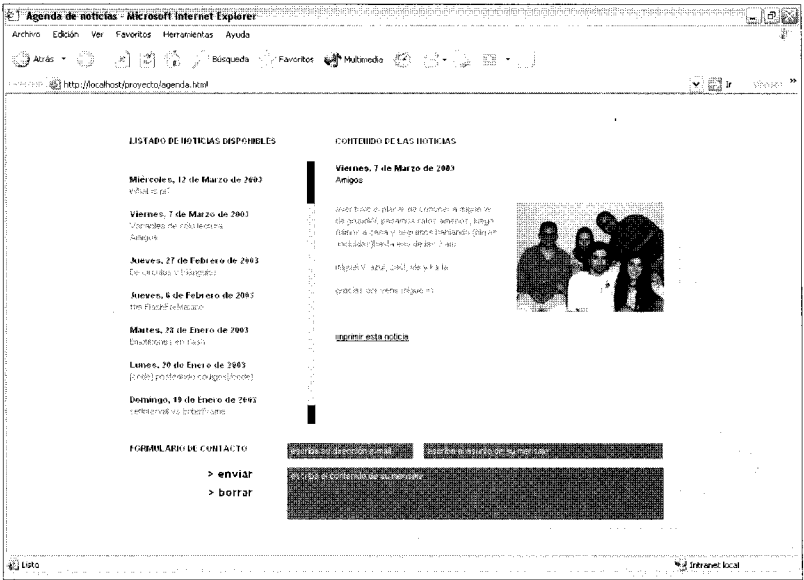


Figura 7.14.
Visualización de noticia con fotografía.

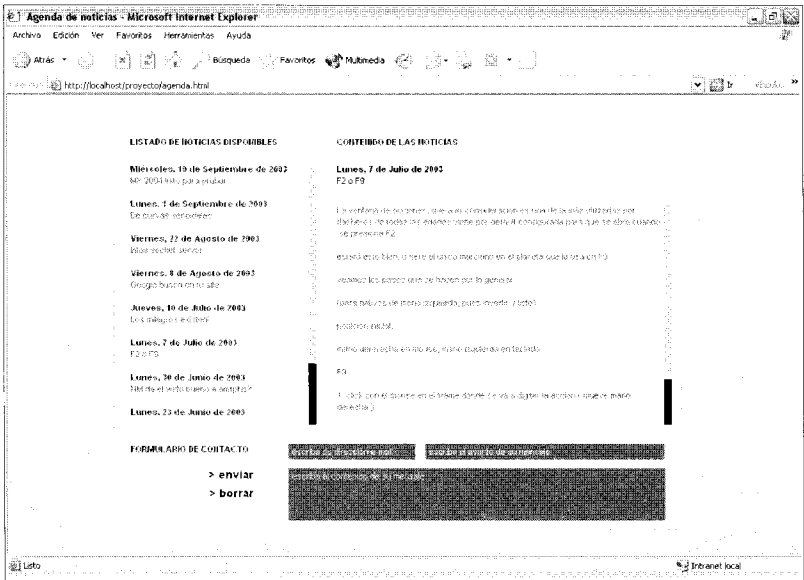


Figura 7.15.
Visualización de noticia sin fotografía.

ventana flotante en la que se muestra, en formato HTML, la fecha, cabecera, contenido y foto de cada una de las noticias, acompañados de dos enlaces a funciones JavaScript: una de ellas cierra la ventana, la otra imprime el documento que se muestra en dicha ventana (figura 7.16). Este documento, al que se aplica la hoja de estilos en cascada que ya utilizamos para el panel de administración, es generado dinámicamente por el documento `imprimir.php` que encontrará en la misma carpeta *proyecto*.

A continuación analizaremos cómo se ha construido la interfaz de la agenda. Abra el documento `material/cap7/proyecto/agenda.flá`, en el que encontrará la disposición de elementos que se muestra en la figura 7.17. Hemos elegido disponer "manualmente" en el escenario los distintos elementos que se utilizarán para acceder a los datos de la agenda (con el propósito de que "visualice" la forma en que se estructurará la información), aunque puede usted crearlos dinámicamente mediante ActionScript.

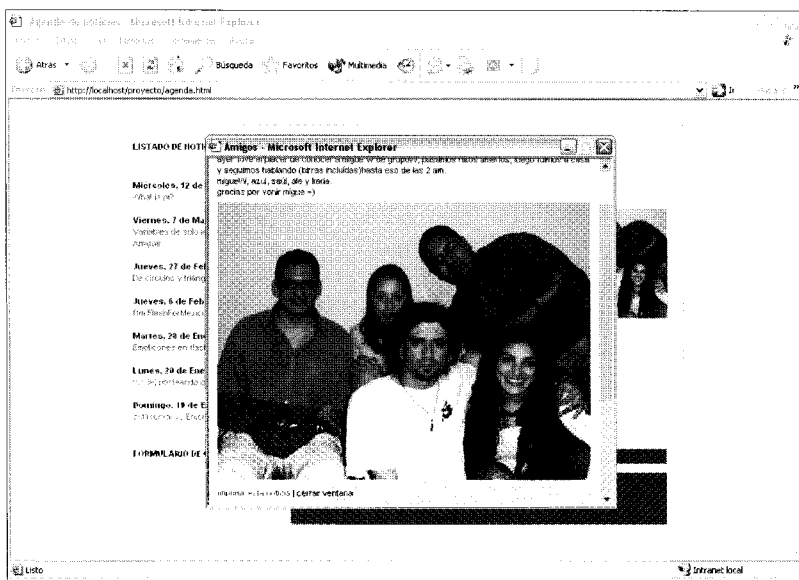


Figura 7.16.
Ventana de impresión de noticias.

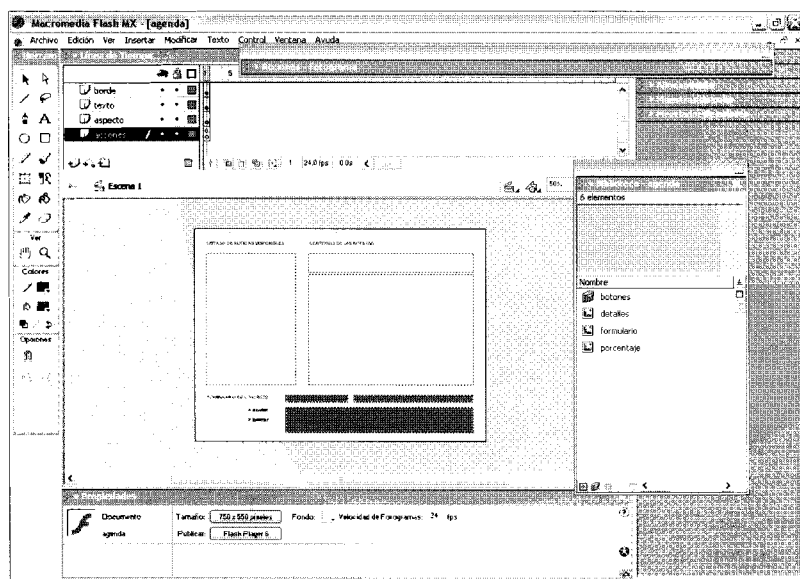


Figura 7.17.
Aspecto general del
documento *agenda.fla*.

En el fotograma 1 de la capa *texto* encontrará, en la parte izquierda, un campo de texto multilinea cuyo nombre de instancia es *noticias* y que está configurado para interpretar texto escrito en código HTML. En la parte derecha encontrará un clip de película cuyo nombre de instancia es *detalles* y que utilizaremos para mostrar los contenidos de las noticias y (si disponen de ella) las fotos que las acompañan. En el interior de este clip existen dos fotogramas: en el primero de ellos encontrará un campo de texto que "ocupa" completamente la parte derecha del escenario, mientras que en el segundo el campo de texto sólo "ocupa" la columna central del escenario. El motivo de desarrollar así este clip de película es el de disponer de dos estructuras para mostrar las noticias dependiendo de si éstas tienen foto o no. Si no tienen foto, el clip de película se quedará en su primer fotograma, mientras que si existe una foto acompañando a la noticia, el clip pasará al segundo fotograma, en el que se ha habilitado un espacio que será utilizado para mostrar las imágenes.

Además de este campo de texto que varía en tamaño dependiendo del fotograma (cuyo nombre de instancia es *contenido*), existe otro en la parte superior del clip de película, cuyo nombre de instancia es *cabecera*, que mostrará la fecha y la cabecera de la noticia.

Por último, en la parte inferior del fotograma 1 de la capa *texto*, encontrará un clip de película cuyo nombre de instancia es *formulario* y que explicaremos posteriormente en este apartado.

Acceda ahora al fotograma 1 de la capa *acciones*, en la que se han descrito las acciones necesarias para controlar la actividad de la agenda. Este es el código que encontrará:

```
// tamaño máximo de las fotos que se utilizarán en la agenda
alturaMaximaFoto=285;
anchuraMaximaFoto=190;

// arrays para los días de la semana y los meses
_global.array_semana=["Domingo","Lunes","Martes","Miércoles","Jueves","Viernes","Sábado","Domingo"];
_global.array_mes=["Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","Agosto","Septiembre","Octubre","Noviembre","Diciembre"];
```

Primero crearemos las variables y estructuras de datos que utilizaremos con posterioridad. Tanto *alturaMaximaFoto* como *anchuraMaximaFoto* son dos valores de referencia que utilizaremos cuando hayamos de añadir una foto a una noticia, para que si supera en anchura o altura uno de estos dos valores podamos redimensionarla con objeto de que se ajuste al tamaño que hemos especificado para mostrarlas. También creamos dos *arrays* globales, uno que almacena los nombres días de la semana y otro que almacena los nombres de los meses del año.

A continuación, declararemos algunas funciones que nos serán de utilidad:

```
// función que retorna la fecha a partir de un string aaaammdd
obtenerFecha=function(texto){
    var textoFecha=texto;
    var dia=new
    Date(textoFecha.substr(0,4),textoFecha.substr(4,2),textoFecha.substr(6,2));
    return (array_semana[dia.getDay()]+", "+dia.getDate()+" de
    "+array_mes[dia.getMonth()]+" de "+dia.getFullYear());
}
```

obtenerFecha es una función que recibe como parámetro una cadena de texto de 8 caracteres en formato "aaaammdd" (tal y como la imprime PHP en el documento XML). Posteriormente se crea una instancia del objeto predefinido de Flash *Date*, inicializándola

con los valores especificados en la fecha (cuatro dígitos para el año, dos dígitos para el mes, dos dígitos para el día) y, haciendo uso de los valores almacenados en los *arrays* globales, se retorna una cadena de texto con el formato "díaSemana, díaMes de Mes de Año".

```
// función que recorta el texto si es superior a 40 caracteres
compruebaTexto=function(texto){
    if (texto.length>40){
        return(texto.substr(0,37)+"...");
    } else {
        return(texto);
    }
}
```

La función `compruebaTexto` se utiliza para mostrar las cabeceras de las noticias en el menú de la izquierda. En algunas ocasiones, el texto de esta cabecera es demasiado largo, por lo que para evitar que ocupe más de una línea mostramos únicamente algunos de sus primeros caracteres unidos a tres puntos suspensivos "..." para indicar que el texto continúa. La función recibe como parámetro el texto a mostrar y evalúa si su longitud es mayor de 40 caracteres. Si no lo es, retorna la cadena de texto que recibió como parámetro; en caso contrario retorna los 37 primeros caracteres del texto concatenados con la cadena "...".

A continuación extenderemos las funcionalidades de algunos de los objetos predefinidos de Flash con métodos propios que vamos a desarrollar. Por ejemplo, el método `isMail` para el objeto *String*, que nos permite averiguar si cualquier cadena de texto (*string*) que invoque a este método es o no una dirección de correo electrónico correcta. Para ello, analiza si en dicha cadena de texto están presentes distintos caracteres que invalidarían el considerarla como dirección de e-mail: si existen los caracteres "@" y "." y si el carácter de arroba está presente antes que el del punto. Si todas las condiciones que especificamos se cumplen, el método retorna el valor 1 (`true`) con lo que podemos considerar que la cadena de texto es una dirección válida. En caso contrario, no lo es, lo que se refleja porque el método retorna el valor 0 (`false`).

```
// extensión del objeto String, método para comprobar e-mail correcto
String.prototype.isMail=function(){
    st=this;
    var mail=1;
    var question,arroba,punto,espacio,ultimo;
    question=st.lastIndexOf("?");
    espacio=st.lastIndexOf(" ");
    arroba=st.lastIndexOf("@");
    punto=st.lastIndexOf(".");
    ultimo=st.charAt(st.length-1);
    if (ultimo!="." and question== -1 and espacio== -1 and arroba!= -1 and punto!= -1
and punto>arroba){
        mail=1;
    } else {
        mail=0;
    }
    return mail;
}
```

A continuación creamos un método para el objeto predefinido de Flash `TextField` para crear un *scroll* que nos permita visualizar el contenido completo de cada uno de los campos de texto que invoquen a este método. Puesto que en nuestra agenda disponemos de campos de texto tanto en la película principal (*noticias*) como dentro de un clip de película (*contenido*, dentro del clip de película cuyo nombre de instancia es *detalles*), comprobaremos a la hora de realizar las acciones si el campo de texto está contenido en la película principal (`_root`, con lo que sabremos que estamos haciendo referencia al campo de texto con nombre de instancia *noticias*) o no lo está (con lo que sabremos que estamos haciendo referencia al campo *contenido* del clip de película *detalles*).

Dependiendo de la película en la que se encuentre el campo de texto, dibujamos un fondo de color negro para el *scroll*, siendo la altura de este fondo la misma que la del campo de texto. Posteriormente creamos otro clip de color blanco (aunque disminuimos el valor de

transparencia mediante su propiedad `_alpha` igual al anterior, y le restamos tantas unidades en altura como líneas no se vean del campo de texto, lo que podemos averiguar si comprobamos el valor de la propiedad `maxscroll` del campo de texto. Si ésta almacena el valor 1, el contenido del campo de texto se visualiza completamente. En caso contrario, la propiedad almacena el número de líneas de contenido que aún faltan por verse en dicho campo.

Este clip blanco, cuya transparencia variará dependiendo de si colocamos o no el puntero del ratón sobre el mismo, será arrastrable entre su posición inicial y la que ocupará cuando su límite inferior coincida con el límite inferior del clip negro. A medida que el clip sea arrastrado, se calcula la diferencia en píxeles desde su posición actual a la inicial, y ese valor es el que se especifica para la propiedad `scroll` del campo de texto, con lo que a medida que el clip blanco se desplaza, también lo hace el contenido que se muestra:

```
// extensión del objeto TextField, creando un scroll para manejar su contenido
TextField.prototype.crearScroll=function(){
    // anchura de la barra de scroll
    var anchura=10;

    // creamos el clip que contendrá el scroll
    // almacenamos el clip en el que está contenido el campo de texto si es
    // distinto de _root
    if (this._parent==_root){
        var clip=_root.createEmptyMovieClip("scroll"+this._name,1);
    } else {
        var clip=detalles.createEmptyMovieClip("scroll"+this._name,1);
        clip.contenedor="detalles";
    }

    // almacenamos el nombre del campo de texto en una variable del nuevo clip
    clip.nombreTextField=this._name;

    // fondo por el que se desplaza la barra de scroll
    clip.createEmptyMovieClip("fondo",1);
    with(clip.fondo){
```

```

        lineStyle(0,0x000000,0);
        beginFill(0x000000,100);
        lineTo(this._x+this._width,this._y);
        lineTo(this._x+this._width+anchura,this._y);
        lineTo(this._x+this._width+anchura,this._y+this._height);
        lineTo(this._x+this._width,this._y+this._height);
        lineTo(this._x+this._width,this._y);
        endFill();
    }
    // barra de scroll
    clip.createEmptyMovieClip("barra",2);
    with(clip.barra){
        lineStyle(0,0x000000,0);
        beginFill(0xFFFFFF,100);
        lineTo(this._x+this._width,this._y);
        lineTo(this._x+this._width+anchura,this._y);
        lineTo(this._x+this._width+anchura,this._y+this._height-this.maxscroll);
        lineTo(this._x+this._width,this._y+this._height-this.maxscroll);
        lineTo(this._x+this._width,this._y);
        endFill();
        _alpha=80;
    }
    // acciones de la barra de scroll
    clip.barra.inicioY=clip.barra._y;
    clip.barra.finalY=clip.barra._y+this.maxscroll;
    clip.barra.inicioX=clip.barra.finalX=clip.barra._x;
    clip.barra.onRollOver=function(){
        this._alpha=85;
    }
    clip.barra.onRollOut=function(){
        this._alpha=80;
    }
    clip.barra.onPress=function(){
        this.startDrag(0,this.inicioX,this.inicioY,this.finalX,this.finalY);

        this.onEnterFrame=function(){
            if (this._parent.contenedor=="detalles"){
                detalles[this._parent.nombreTextField].scroll=(this._y-
this.inicioY);
            } else {
                _root[this._parent.nombreTextField].scroll=(this._y-
this.inicioY);
            }
        }
    }
    clip.barra.onRelease=function(){
        delete this.onEnterFrame;
        this.stopDrag();
    }
    clip.barra.onReleaseOutside=clip.barra.onRelease;
}

```

La siguiente función se utiliza en caso de que la imagen que se ha de utilizar para complementar una noticia supere en anchura y/o altura las dimensiones que se especificaron en las variables declaradas al comienzo de este fotograma. Recibiendo como parámetros la anchura y altura de la imagen, se calcula si supera las dimensiones establecidas, lo que provoca que se redimensione al valor necesario la propiedad cuyo valor supera su "límite". Posteriormente, y ya que se ha redimensiona-

do una de las propiedades, hay que hacer lo mismo con la otra para que la imagen no se deforme, para lo que se calcula un pequeño coeficiente que establece la relación entre el tamaño original de la propiedad que se modificó y su tamaño actual. Posteriormente, este coeficiente se aplica a la otra propiedad, con lo cual nos aseguramos de que el tamaño de la fotografía se adaptará al espacio que hemos reservado para ella, devolviendo la función un objeto con las dos propiedades "ajustadas":

```
// función que sirve para escalar una imagen
// recibe dos parámetros (anchura y altura a escalar)
// retorna un objeto con dos atributos (anchura y altura escalada para la
imagen)
//
//  datos=escalarImagen(285,326);
//  trace(data.alto)
//  trace(data.anch)
escalarImagen=function(alto, ancho){
  if (alto>alturaMaximaFoto){
    if (ancho<=anchuraMaximaFoto){
      coeficiente=alto/alturaMaximaFoto;
      altura=alturaMaximaFoto;
      anchura=ancho/coeficiente;
    } else {
      coeficiente=alto/alturaMaximaFoto;
      altura=alturaMaximaFoto;
      anchura=ancho/coeficiente;
      if (anchura>anchuraMaximaFoto){
        coeficiente=ancho/anchuraMaximaFoto;
        anchura=anchuraMaximaFoto;
        altura=alto/coeficiente;
      }
    }
  } else {
    if (ancho>anchuraMaximaFoto){
      coeficiente=ancho/anchuraMaximaFoto;
      anchura=anchuraMaximaFoto;
      altura=alto/coeficiente;
    }
  }
  return ({anch:anchura, alto:altura});
}
```

La función `imprimir` recibe como parámetro un valor de identificador único (`id`) de cada noticia y lo envía mediante el método `GET` para abrir en una ventana flotante el documento `imprimir.php` que nos permitirá imprimir dicha noticia:

```
// función que sirve para abrir el pop-up de impresión:
imprimir=function(datos){

getURL("javascript:MM_openBrWindow('imprimir.php?id="+datos+"','imprimir','toolbar=no,location=no,
status=no,menubar=no,scrollbars=yes,resizable=no','520','450','true')");;
}
```

La siguiente función es llamada cuando se produce la pulsación de una noticia. Al estar los enlaces de las mismas dentro de texto formateado con HTML, hemos de hacer uso de la función `asfunction` de que dispone Flash para desencadenar acciones con enlaces de ese tipo. La utilización de esta función puede explicarse con este ejemplo simple:

```
<a href='asfunction: saludo,
parametro1'>saludar</a>
```

Cuando queremos realizar una llamada a una función desde un enlace HTML especificamos primeramente el nombre de la función y, separado por una coma, un parámetro, con lo que si dispusiéramos en nuestra película Flash de una función como ésta:

```
saludo=function(datos){
trace(datos);
}
```

El realizar la llamada a la función de la siguiente forma:

```
<a href='asfunction: saludo,
hola!'>saludar</a>
```

produciría que cuando pulse el enlace "saludar" se mostraría el texto "hola!" en la ventana de Salida de Flash.

El problema que se da es el de que sólo se puede incluir un parámetro en la llamada a la función, con lo que si deseamos enviar más información hemos de buscar una alternativa para ello.

Una solución que utilizamos frecuentemente es la de escribir los distintos parámetros que deseamos enviar a la función como uno solo en una cadena de texto, separándolos por juegos de caracteres que consideremos especiales, por ejemplo "---":

```
<a href='asfunction: saludo, dato1-
--dato2---dato3---dato4'>saludar</a>
```

Posteriormente, al definir la función que es llamada por `asfunction`, podemos utilizar el método `split` del objeto predefinido de Flash *String*, que nos permite separar una cadena en base a un carácter o juego de caracteres de referencia, retornando los resultados como elementos de un *array*. Por ejemplo, si dispusiéramos de esta cadena de texto:

```
datos="dato1---dato2---dato3---dato4";
```

Utilizaríamos `String.split` para extraer los distintos datos:

```
mis_datos=datos.split("---");
```

A partir de este momento, `mis_datos` es un *array* que contiene cuatro celdas:

- `mis_datos[0]` almacena la cadena de texto "dato1".
- `mis_datos[1]` almacena la cadena de texto "dato2".
- `mis_datos[2]` almacena la cadena de texto "dato3".
- `mis_datos[3]` almacena la cadena de texto "dato4".

De esta forma podemos enviar cuatro parámetros a la función "camuflados" en una sola cadena de texto. Esto es precisamente lo que haremos en la función `obtenerDetalle`, que "recibe" los parámetros `id`, `dia`, `cabece-ra` y `foto` separados por la cadena de texto "---". Si la variable que almacena el nombre de la foto (`datos.split("---")[3]`) contiene un texto distinto de "n/a", entonces ya disponemos de la ruta para la imagen, con lo que enviamos el clip de película *detalles* al fotograma 2 y allí creamos dinámicamente un clip de película vacío que utilizaremos para cargar la fotografía.

Tras crear un pequeño cargador porcentual para la imagen, comprobamos si su tamaño en anchura y altura está dentro de los límites especificados por las variables `alturaMaximaFoto` y `anchuraMaximaFoto`. En caso de que sobrepase dichos límites, realizamos una llamada a la función `escalarImagen` con el tamaño actual de la imagen, el cual ajustamos con los valores que retorna la función.

En caso de que la noticia no disponga de fotografía enviamos el clip *detalles* al fotograma 1 y comprobamos si existe el clip que almacena las fotos (de alguna noticia anterior) para eliminarlo utilizando el método `removeMovieClip` del objeto predeterminado de Flash *MovieClip*.

Finalmente, damos contenido al campo de texto *cabecera* del clip de película *detalles* (utilizando `datos.split("---")[1]`, que contiene la fecha y que pasamos previamente por la función `obtenerFecha` para disponer de ella en el formato que explicamos en la declaración de la función); y por último, creamos un objeto *LoadVars* al que especificamos la variable `id` (contenida en `datos.split("---")[0]`) para realizar una llamada al documento `contenidos.php`, que devolverá en la variable *contenido* el texto de la noticia para mostrarlo en el campo de texto *contenido* del clip de película *detalles*. Añadimos también un enlace con el texto "imprimir esta noticia" que llamará a la función `imprimir` pasándole como parámetro el contenido de `datos.split("---")[0]` (el identificador único `id`).

Comprobamos el valor de la propiedad `maxscroll` del campo de texto una vez incorporado el texto de la noticia y creamos un *scroll* para el mismo invocando al método `crearScroll` del objeto *TextField* (que hemos explicado con anterioridad).

```

// función que se activa con la pulsación de una noticia
obtenerDetalle=function(datos){
    if (datos.split("---")[3]!="n/a"){
        detalles.gotoAndStop(2);

        detalles.createEmptyMovieClip("foto",10);
        detalles.foto._x=detalles.contenido._x+detalles.contenido._width+20;
        detalles.foto._y=detalles.contenido._y;

        // carga de la foto, tamaño máximo 285 x 190 (anchura x altura)
        // si es mayor en anchura o altura se redimensiona para que se ajuste
        // utilizando la función escalarImagen()
        detalles.foto.loadMovie(datos.split("---")[3]);

        // creación de un campo de texto para calcular el porcentaje cargado de la foto
        detalles.attachMovie("porcentaje","porcentaje",11,{_x:detalles.foto._x,_y:detalles.foto._y});
        detalles.onEnterFrame=function(){
            var cargado=detalles.foto.getBytesLoaded();
            var total=detalles.foto.getBytesTotal();
            if (!isNaN(cargado/total)){
                detalles.porcentaje.valor.text=(100*Math.round(cargado/total))+ " %";
            }

            if (detalles.foto._width!=0 and cargado==total){
                if (detalles.foto._height>alturaMaximaFoto or
                detalles.foto._width>anchuraMaximaFoto){
                    var
nuevoTamano=escalarImagen(detalles.foto._height,detalles.foto._width);
                    detalles.foto._width=nuevoTamano.ancha;
                    detalles.foto._height=nuevoTamano.alto;
                }

                detalles.porcentaje.removeMovieClip();
                delete detalles.onEnterFrame;
            }
        }
    } else {
        detalles.gotoAndStop(1);
        if (detalles.foto) detalles.foto.removeMovieClip();
    }

    detalles.cabecera.htmlText="<b>"+obtenerFecha(datos.split("---")[1])+"</b><br>"+datos.split("---")[2];
    detalles.contenido.htmlText="";

    var lv=new LoadVars();

    lv.onLoad=function(exito){
        if (exito){
            if (this.output=="ok"){
                detalles.contenido.htmlText+=this.contenido;

                // opción de impresión de los contenidos
                detalles.contenido.htmlText+=("<br><br><br><font
color='#000000'><u><a href='asfunction:_root.imprimir,'"+datos.split("---")
[0]+">imprimir esta noticia</a></u></font>");

```

```

        detalles.scrollcontenido.removeMovieClip();
        if (detalles.contenido.maxscroll!=1)
detalles.contenido.crearScroll();
        } else {
            detalles.contenido.htmlText+=("Error:  "+this.msg);
        }
    } else {
        detalles.contenido.htmlText+="Error al cargar los datos";
    }
}
lv.id=datos.split("---")[0];
lv.sendAndLoad("http://localhost/proyecto/contenidos.php",lv,"POST");

detalles.contenido.scroll=1;
}

```

Creamos un objeto XML que cargará las noticias desde el documento `agenda.xml` y establecemos la propiedad `ignoreWhite` de dicho objeto con el valor `true` para que Flash no interprete los saltos de línea como nuevos nodos:

```

// objeto XML que recibe el listado de noticias
datos_agenda=new XML();
datos_agenda.ignoreWhite=1;

```

Y definimos el evento `onLoad` para el objeto, cuyas acciones se ejecutarán cuando se hayan cargado completamente los datos de `agenda.xml`.

Se accede al primer nodo (*agenda*) y posteriormente a cada uno de sus nodos hijos (*noticia*) en los que se lee el atributo `dia` y en los que se recorre cada uno de los nodos *datos*, obteniendo la información de los distintos atributos (`id`, `cabecera` y `foto`) y formateándola para que sea mostrada en el campo de texto *noticias*.

Observe cómo se utiliza `asfunction` para realizar una llamada a la función `obtenerDetalle` pasando como parámetro una cadena de texto que contiene los contenidos de los atributos del nodo.

Por último se comprueba la propiedad `maxscroll` del campo de texto y se le añade un `scroll` en caso de que su valor sea distinto de 1:

```
// función que procesa la información de la lista de noticias
datos_agenda.onLoad=function(exito){
    if(exito){
        var texto_noticias="";
        var raiz=this.firstChild;
        var hijos=this.firstChild.childNodes.length;
        if (hijos!=0){
            for(var n=0;n<hijos;n++){
                texto_noticias+=("<font
color='#333333'><b>" + obtenerFecha(raiz.childNodes[n].attributes.dia) + "</b></
font><br>");
                for(var m=0;m<raiz.childNodes[n].childNodes.length;m++){
                    texto_noticias+=("<a href='asfunction:obtenerDetalle,
"+raiz.childNodes[n].childNodes[m].attributes.id+"---
"+raiz.childNodes[n].attributes.dia+"---
"+raiz.childNodes[n].childNodes[m].attributes.cabecera+"---"
+raiz.childNodes[n].childNodes[m].attributes.foto+"'">
"+compruebaTexto(raiz.childNodes[n].childNodes[m].attributes.cabecera) + "</
a><br>");
                }
                if (n<hijos-1) texto_noticias+= "<br>";
            }
            noticias.htmlText=texto_noticias;
            if (noticias.maxscroll!=1) noticias.crearScroll();

            detalles.cabecera.htmlText="Elija una noticia en el menú de la
izquierda.";
        } else {
            noticias.text="No hay noticias disponibles."
        }
    } else {
        noticias.htmlText="<b>Error al cargar los datos</b>";
    }
}
```

Por último, procedemos a la carga del documento XML utilizando una variable aleatoria (rn) para asegurarnos de que no se carga ninguna versión de agenda.xml que estuviera presente en la memoria caché de nuestro navegador:

```
// acceso al documento XML que contiene el listado de noticias
datos_agenda.load("http://localhost/proyecto/agenda.xml?rn=" + new
Date().getTime());
```

A continuación explicaremos el funcionamiento del formulario, que dispone de tres campos de introducción de texto dentro del clip de película *formulario*, *remite*nte, *asunto* y *mensaje*. La particularidad de estos últimos es

la de inicializarse con los valores declarados como textos iniciales en *texto_remitente*, *texto_asunto* y *texto_mensaje*, y el hecho de que una vez que seleccionemos el campo de texto para escribir en él (lo que se

detecta por el evento *onSetFocus*) borramos su contenido si éste coincide con el de su texto inicial. De igual forma, cuando se selecciona otro campo de texto (lo que el campo activo hasta entonces detecta mediante el evento

onKillFocus), el campo que estaba activo recupera su texto inicial si está vacío.

El botón **borrar** establece el texto inicial para cada uno de los campos de texto, mientras que el botón **enviar** comprueba si el texto escrito en el campo de texto remitente es una dirección de correo electrónico correcta (para lo que utiliza el método `isMail` que hemos declarado para el objeto *String*).

Si es una dirección correcta, se realiza una llamada al documento `contacto.php` (que explicaremos a continuación) y se envía el clip *formulario* al fotograma *enviar* para notificar que el envío se ha realizado con éxito (figura 7.18). En caso contrario, se avisa al usuario de que su dirección es incorrecta enviando el clip *formulario* al fotograma *rellenar* (figura 7.19).

```
// -----  
//  
// acciones y variables del formulario  
  
// textos iniciales  
texto_remitente="escriba su dirección e-mail";  
texto_asunto="escriba el asunto de su mensaje";  
texto_mensaje="escriba el contenido de su mensaje";  
  
formulario.remitente.text=texto_remitente;  
formulario.remitente.onSetFocus=function(){  
    if (this.text==texto_remitente) this.text="";  
}  
formulario.remitente.onKillFocus=function(){  
    if (this.text=="") this.text=texto_remitente;  
}  
  
formulario.asunto.text=texto_asunto;  
formulario.asunto.onSetFocus=function(){  
    if (this.text==texto_asunto) this.text="";  
}  
formulario.asunto.onKillFocus=function(){  
    if (this.text=="") this.text=texto_asunto;  
}  
  
formulario.mensaje.text=texto_mensaje;  
formulario.mensaje.onSetFocus=function(){  
    if (this.text==texto_mensaje) this.text="";  
}  
formulario.mensaje.onKillFocus=function(){  
    if (this.text=="") this.text=texto_mensaje;  
}  
  
formulario.borrar.onRelease=function(){  
    formulario.remitente.text=texto_remitente;  
    formulario.asunto.text=texto_asunto;  
    formulario.mensaje.text=texto_mensaje;  
}  
formulario.enviar.onRelease=function(){  
    if (formulario.remitente.text.isMail()){  
        var datos=new LoadVars();  
        datos.remitente=formulario.remitente.text;
```

```

    datos.asunto=formulario.asunto.text;
    datos.mensaje=formulario.mensaje.text;
    datos.sendAndLoad("contacto.php",datos,"POST");
    formulario.gotoAndPlay("enviar");
    formulario.borrar.onRelease();
} else {
    formulario.gotoAndPlay("rellenar");
}
}

```



Figura 7.18.

Notificación de envío correcto de los datos del formulario.

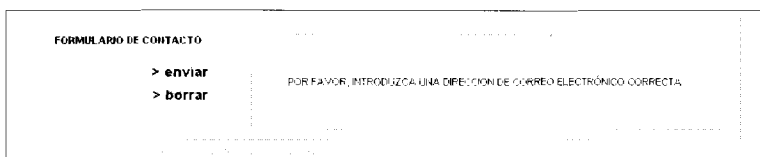


Figura 7.19.

Notificación de error al intentar enviar los datos del formulario.

Una vez explicada la realización de la interfaz en Flash de la agenda, terminaremos este capítulo analizando los documentos que retornan los contenidos de cada noticia y los que se utilizan para imprimirlas.

Muestra de contenidos

El documento `contenidos.php` es el que retorna el texto de una noticia cuando Flash envía el identificador `id` de la misma. Para ello comprueba la existencia (línea 2) de la variable `id` (enviada mediante el método `POST`). Si existe, se incluyen los documentos de configuración (líneas 3 y 4), establecemos la conexión con la base de datos almacenando el apuntador en la variable `$cnx` (línea 6) y almacenando en la variable `$res` el resultado de la consulta SQL, que consiste en seleccionar la columna `texto` de la tabla `agenda` allí donde el valor de `id` del registro coincida con

el de la variable `id` enviada mediante el método `POST` (línea 7).

Comprobamos el número de filas en el resultado (línea 8) para que, si dicho número es distinto de cero, asignar el valor de la única columna en el resultado (`texto`) a la variable `$contenido` utilizando la función `list` (línea 10). En la línea 11 se imprimen los resultados en formato de texto codificado UTF-8 y en la línea 14 se imprime un mensaje de error si no existen resultados, para cerrar finalmente la conexión con MySQL en la línea 17.

Si la variable `id` no existe o no ha sido enviada utilizando el método `POST`, mostramos un mensaje de error en pantalla (líneas 18-21). Al igual que hicimos en su momento, declaramos también una variable de nombre `output` para comunicarle a Flash que todo ha salido bien (almacenando la cadena "ok") o que se ha producido algún fallo (almacenando la cade-

na "error"), añadiendo entonces la variable msg con el mensaje de error que se haya producido.

El código completo de este documento (que puede encontrar en la carpeta material/cap7/proyecto del CD) es el siguiente:

```
1<?
2if(!empty($_POST['id'])) {
3    include("includes/config.php");
4    include("includes/funciones.php");
5    //nos conectamos a la bd.
6    $cnx = conectar();
7    $res = mysql_query("SELECT texto FROM agenda WHERE id = '".$_POST['id']."' ) or
die("&output=error&msg=".mysql_error());
8    if( mysql_num_rows($res) > 0 ) {
9        //si hay datos.
10        list($contenido) = mysql_fetch_array($res);
11        echo "&output=ok&contenido=".utf8_encode($contenido);
12    } else {
13        //no hay datos.
14        echo "&output=error&msg=no hay información&";
15    }
16    //cerramos la conexión con mysql.
17    mysql_close($cnx);
18 } else {
19     //no se pasó el id.
20     echo "&output=error&msg=no hay información&";
21 }
22?>
```

Impresión en papel de datos

Para imprimir en papel los contenidos de cada noticia vamos a utilizar dos ficheros: imprimir.php y plantilla-imprimir.php. El primero de ellos revisa la existencia de la variable id enviada mediante el método GET para realizar una consulta seleccionando los datos de la noticia. Una vez realizada esta tarea, incluirá el documento plantilla-imprimir.php, que es la página que muestra la información.

Analizamos primero el código del documento imprimir.php: en caso de existir resultados tras la consulta SQL (línea 9), la información de las columnas se guarda en las variables \$cabecera, \$texto, \$fecha y \$foto, utilizando para ello la función list de PHP.

Posteriormente (línea 13) se le da formato a la fecha haciendo uso de la función formatFecha para guardar la fecha en la base de datos con el formato "aaaammdd", tal y como explicamos al analizar el documento funciones.php de este mismo capítulo.

La función nl2br sirve para sustituir los saltos de línea por etiquetas
 de HTML, mostrando el texto correctamente en la página de imprimir; en caso de no haber datos, mostramos un mensaje de error (líneas 18-21) y, una vez que disponemos de los datos del resultado, incluimos la plantilla para imprimir, que es la que mostrará los resultados con formato en la pantalla. Cerramos la conexión con MySQL (línea 23) y redireccionamos a la página principal en caso de que no exista la variable id (líneas 27 y 28).

El código completo para este documento (que puede encontrar en la carpeta material/cap7/proyecto del CD) es el siguiente:

```
1<?
2if(!empty($_GET['id'])){
3    include("includes/config.php");
4    include("includes/funciones.php");
5    //nos conectamos a la bd.
6    $cnx = conectar();
7
8    $res = mysql_query("SELECT cabecera,texto,fecha,foto FROM agenda WHERE id =
9".$_GET['id']) or die("&output=error&msg=".mysql_error());
10    if( mysql_num_rows($res) > 0 ){
11        //si hay datos.
12        list($cabecera,$texto,$fecha,$foto) = mysql_fetch_array($res);
13        //formateamos la fecha.
14        $fecha = formatFecha($fecha);
15        //cambiamos los \n por <br> para html
16        $texto = nl2br($texto);
17        include ("includes/plantilla-imprimir.php");
18        exit;
19    }else{
20        //no hay datos.
21        echo "error al buscar el registro.";
22    }
23    //cerramos la conexion con mysql.
24    mysql_close($cnx);
25}
26header ("Location: index.html");
27exit;
28?>
```

La inclusión de documentos tras comprobar la existencia de determinadas variables (tanto las enviadas mediante el método GET como las del resultado) nos ayuda desde el punto de vista de que podemos diseñar la página como si fuera un documento HTML común, con la salvedad de que incluimos distintas etiquetas PHP, ya sea para verificar la existencia de variables o para realizar la impresión de las mismas.

El documento `plantilla-imprimir.php` comienza comprobando la existencia de la variable `id` y que haya sido enviada mediante el método GET (líneas 2-4). Posteriormente

hacemos uso de las variables de que se dispone (`$cabecera`, `$texto`, `$fecha` y `$foto`) para personalizar la página. En la línea 9 asignamos el título a la página utilizando el texto contenido en la variable `$cabecera`, también utilizado en la línea 16 junto con la fecha de la noticia. El texto es impreso en la línea 20.

Revisamos también el contenido de la variable `$foto` para saber si la noticia dispone de una imagen asociada; si es así, se imprime la celda junto con la etiqueta de imagen que la contendrá (líneas 22-31).

Por último, se incluyen dos enlaces a funciones de JavaScript, uno para la impresión del contenido de la ventana del navegador y otro para cerrar la misma (línea 33).

El código completo para este documento (que puede encontrar en la carpeta `material/cap7/proyecto/includes` del CD) es el siguiente:

```
1<?
2if(empty($_GET['id'])) {
3    exit;
4}
5?>
6<html>
7<head>
8<title>
9<? echo $cabecera;?>
10</title>
11<link href="estilos.css" rel="stylesheet" type="text/css">
12</head>
13<body bgcolor="#DDDDDD" marginwidth="0" marginheight="0" topmargin="0"
leftmargin="00">
14<table width="450" height="350" border="0" align="center" cellpadding="10"
cellspacing="0">
15    <tr height="30">
16        <td valign="top" class="cabecera"> <span class="cabeceraBold"><? echo
$fecha;?></span><br>
17        <? echo $cabecera;?></td>
18    </tr>
19    <tr>
20        <td valign="top" class="texto"><?echo $texto?><br></td>
21    </tr>
22    <?
23    if($foto != "N/A" && !empty($foto)) {
24        //hay foto.
25    ?>
26    <tr>
27        <td align="center"> "> </td>
28    </tr>
29    <?
30    } //fin if de foto
31    ?>
32    <tr height="30">
33        <td valign="top" class="pie"> <a href="javascript:window.print()"> imprimir
esta noticia</a> | <a href="javascript:window.close()">cerrar ventana</a>
34    </td>
35    </tr>
36</table>
37</body>
38</html>
```

Zona de contacto

Para el envío del formulario de la agenda a una dirección de correo electrónico utilizamos el documento `contacto.php`, haciendo uso de la función de PHPmail, pero, anteriormente, daremos formato al correo utilizando las variables enviadas desde Flash (`remite`nte, `asunto` y `mensaje`).

Comenzaremos revisando la existencia de las variables del formulario realizado en Flash (línea 2); en nuestro caso revisamos la existencia de la variable `remite`nte, pero si lo desea puede comprobar que existan las tres variables (`remite`nte, `asunto` y `mensaje`).

Posteriormente, almacenamos la fecha actual en la variable `$time`, utilizando la función `date` (línea 3) y establecemos, dentro de la cabecera del correo, el valor contenido en la

variable remitente como persona que envía el correo electrónico, utilizando para ello "From: " junto con el valor contenido en la variable (línea 4).

Las líneas 6-12 sirven para asignarle formato al correo, almacenando el texto de lo que será el cuerpo del correo en la variable `$cuerpo` y creamos una variable llamada `$to_email`, que es la dirección de correo a la que deseamos que llegue la información tecleada en el formulario (que usted deberá personalizar si decide hacer uso de este documento en un caso real, línea 14).

Finalmente, hacemos uso de la función `mail` de PHP para enviar el formulario utilizando las variables que hemos especificado como parámetros de la función.

El código completo para este documento (que puede encontrar en la carpeta `material/cap7/proyecto del CD`) es el siguiente:

```
1<?php
2if($_POST['remite']){
3    $time = date("l dS of F Y h:i:s A");
4    $headers="From: $remite\n";
5    $cuerpo="Llenaron el formulario";
6    $cuerpo.="esta es la información:\n";
7    $cuerpo.="-----\n";
8    $cuerpo.="email: $remite\n";
9    $cuerpo.="Asunto: $asunto\n";
10   $cuerpo.="mensaje:\n $mensaje\n\n";
11   $cuerpo.="fecha : $time\n";
12   $cuerpo.="-----\n";
13   // envia el email
14   $to_email="micorreo@miserver.com";
15   mail($to_email,$motivo,$cuerpo,$headers);
16}
17?>
```

En este punto finaliza la construcción de la agenda que hemos desarrollado como ejemplo práctico para el libro. En la carpeta `material/cap7/proyecto/fuente-noticias` del CD encontrará los materiales (tanto textuales como fotográficos) que hemos utilizado como información para almacenar en la agenda.

En el próximo capítulo encontrará una selección de enlaces en los que podrá conseguir recursos y tutoriales tanto de Flash como de PHP/MySQL para complementar los conocimientos adquiridos con la lectura de este libro.

```
gotoAndPlay("capitulo_8");
```

Capítulo 8

Recursos y enlaces

En este capítulo haremos referencia a distintos sitios Web y recursos bibliográficos que creemos pueden serle de utilidad para trabajar tanto con Flash como con PHP/MySQL:

- **Flash, PHP y MySQL: Contenidos dinámicos**

<http://www.granatta.com/lib/flashphp>
(en castellano)

Sitio Web del libro que tiene en sus manos. Información sobre este libro, así como de los autores y sus trabajos.

- **ActionScript.com**

<http://www.actionscript.com> (en inglés)

Comunidad dedicada a proveer de noticias e información relevante para desarrolladores de Flash.

- **Actionscript.org**

<http://www.actionscript.org> (en inglés)

Comunidad dedicada a noticias e informaciones relevantes para desarrolladores Flash.

- **Alesys**

<http://www.alesys.net> (en castellano)

Sitio Web personal de Rolf Ruiz, en el que encontrará foros de apoyo dedicados a Flash, documentos .fla para descargar, así como tutoriales en formato .pdf.

- **AMF PHP**

<http://www.amfphp.org> (en inglés)

Sitio Web en el que encontrará el modo de implementar el funcionamiento de Flash Remoting utilizando PHP.

- **AS Nativos**

<http://www.sidedev.net/asnativos> (en castellano)

Lista de correo dedicada a desarrolladores de Actionscript en castellano.

- **BomberStudios**

<http://www.bomber-studios.com>
(en inglés)

Blog personal de Ale Muñoz, de Sevilla, en el que encontrará noticias relacionadas con Flash y su aplicación no sólo en Web sino también en otros dispositivos.

- **E-Site, cuadernos de diseño y creación de websites**

<http://www.e-site-es.com> (en castellano)

Revista dedicada exclusivamente a Flash, en castellano y dirigida por Salvador Cuenca.

- **EditPlus**

<http://www.editplus.com>

Este editor de texto reconoce la mayoría de los lenguajes de programación y Web, siendo fácil la incorporación de otros nuevos.

- **Flash Argentina**

<http://www.flashargentina.com.ar>
(en castellano)

Comunidad Flash argentina en la que podrá encontrar documentos .fla para descargar, además de foros de soporte para sus usuarios.

- **Flash Latino**

<http://www.flashla.com> (en castellano)

Fundado por Carlos Zumbado (Costa Rica) y Santiago Anglés (Uruguay), cuenta con unos magníficos foros sobre prácticamente todo lo relacionado con Flash y una extensa lista de colaboradores moderándolos.

- **Flash-es**

<http://www.flash-es.net> (en castellano)

Portal decano de Flash en castellano. Artículos y tutoriales para aprender desde cero no sólo acerca de Flash, sino también Director, Dreamweaver, etc.

- **Flashcoders**

<http://chattyfig.figleaf.com> (en inglés)

Lista de correo dedicada a desarrolladores de Actionscript.

- **FlashKit**

<http://www.flashkit.com> (en inglés)

Uno de los mayores portales de recursos para descargar documentos .fla (prima la cantidad sobre la calidad), sonidos, imágenes o debatir en sus foros.

- **Flashmaestro**

<http://www.flashmaestro.fm>
(en castellano)

Durante mucho tiempo fue exclusivamente una lista de correo (de tráfico bastante elevado), servicio que ha sido añadido a su actual aspecto de portal donde encon-

trar tutoriales, foros, enlaces y documentos fuente.

- **FlashXL**

<http://www.flashxl.com> (en castellano)

Comunidad repleta de enlaces, tutoriales y archivos para descargar.

- **Flazoom**

<http://www.flazoom.com> (en inglés)

Noticias y enlaces relacionadas con la usabilidad en Flash.

- **Guía Práctica: ActionScript para Flash MX**

<http://www.granatta.com/lib/flashmxactionscript> (en castellano)

Sitio Web del libro sobre ActionScript para Flash MX escrito por Daniel de la Cruz Heras. Información sobre este libro, así como del autor y sus trabajos.

- **Kadazuro**

www.kadazuro.com (en castellano)

Sitio Web personal de Carlos Zumbado. Infinidad de tutoriales sobre Flash, en solitario, o combinado con PHP.

- **Kenike**

<http://www.kenike.org> (en castellano)

Sitio personal de Dragan Ardala, noticias y trucos relacionados con Flash actualizados frecuentemente.

- **Macromedia Designers and Developers Center**

<http://www.macromedia.com/desdev> (en inglés)

Centro de diseñadores y desarrolladores de Macromedia. Actualizado constantemente con nuevos tutoriales y actualizaciones de productos de la compañía.

- **Colin Moock**

<http://www.moock.org> (en inglés)

Sitio Web personal de Colin Moock, en el que encontrará tutoriales, trucos e información sobre sus libros y artículos.

- **Nomaster**

<http://www.nomaster.com> (en castellano)

Sitio Web que ofrece noticias y multitud de documentos .fla para descargar. Organizan concursos frecuentemente y sus foros son realmente didácticos.

- **OneLX**

<http://www.onelx.com> (en varios idiomas)

Sitio Web personal de Fernando Flórez, en el que también podrá encontrar el *blog* que mantiene junto con Santiago Anglés (www.sangles.com), frecuentemente actualizado con nuevos componentes y noticias relacionadas con Flash.

- **Santiago Ortiz**

<http://www.moebio.com/santiago>
(en castellano)

Sitio Web personal de Santiago Ortiz, en el que encontrará experimentos con las matemáticas y/o el sonido como tema principal.

- **PHP Builder**

<http://www.phpbuilder.com> (en inglés)

Sitio Web en el que encontrará noticias, artículos y tutoriales de PHP.

- **PHP NET**

<http://www.php.net> (en varios idiomas)

Sitio Web de referencia de PHP en Internet.

- **PHP Object**

<http://ghostwire.com/resources/phpobject/> (en inglés)

Alternativa a Flash Remoting para desarrolladores de PHP.

- **PHP Resources Index**

<http://php.resourcesindex.com>
(en inglés)

Sitio Web en el que podrá encontrar más de dos mil aplicaciones, módulos y *scripts* de PHP.

- **Kali Romiglia**

<http://www.romiglia.com> (en castellano)

Sitio Web personal de Kali Romiglia, en el que podrá encontrar información sobre sus trabajos y su libro *Manual avanzado de Flash MX*, así como descargar su *pack* de componentes esenciales.

- **Scite Flash**

<http://www.bomberstudios.com/scite-flash/>

Sitio Web oficial de Scite Flash, un editor de ActionScript basado en Scintilla creado por Ale Muñoz, que dispone de numerosas características que lo hacen idóneo para utilizarlo conjuntamente con Flash.

- **SourceForge**

<http://sourceforge.net> (varios idiomas)

Sitio Web en el que encontrará más de sesenta mil proyectos de código abierto (*Open Source*).

- **Subflash**

<http://www.subflash.com> (en castellano)

Comunidad que cuenta con tutoriales, foros y documentos .fla de gran calidad para descargar.

- **Tutoflash**

<http://www.tutoflash.com> (en castellano)

Tutoriales, tutoriales y más tutoriales en este sitio Web.

- **Ultrashock**

<http://www.ultrashock.com> (en inglés)

Quizá el portal sobre Flash más completo, por cantidad y calidad en los ficheros disponibles, por sus noticias y por la lista de moderadores de sus foros.

- **Webmonkey**

<http://hotwired.lycos.com/webmonkey/programming/php/> (en inglés)

Sitio Web en el que encontrará tutoriales y artículos sobre PHP/MySQL.

- **Were-Here**

<http://www.were-here.com> (en inglés)

Conocido por sus magníficos foros de discusión, también ofrece artículos y reportajes relacionados con Flash.

- **X-Flash**

<http://xflash.8k.com> (en castellano)

En este completo sitio Web encontrará tutoriales, artículos y ficheros fuente para descargar.

Apéndice

Contenido del CD-ROM

Este CD-ROM es una versión dual para Windows y Macintosh en el que, además de los archivos que se han utilizado a lo largo de este libro, también encontrará el material necesario para ponerlos en funcionamiento y versiones de evaluación de diversas herramientas de diseño que son de gran utilidad para cualquier usuario.

A continuación detallaremos el contenido de las distintas carpetas que encontrará en el CD-ROM:

- **Material:** Contiene, ordenados por capítulos, todos los documentos fuente de Flash, PHP y MySQL incluidos en el libro y mediante los cuales se explican sus contenidos, entre los que se aclara cómo utilizar los documentos de esta carpeta. Dentro de la misma encontrará también otra carpeta de nombre *Instaladores*, sólo para PC, en la que encontrará la aplica-

ción *AppServ*, que le permitirá instalar de forma rápida y cómoda el software necesario para gestionar sus bases de datos y utilizar su ordenador como servidor que le permita realizar pruebas y ejercicios sin necesidad de conectarse a Internet.

- **Software Macromedia:** Contiene versiones de evaluación (*trials*) de programas de Macromedia, que podrá utilizar para trabajar con los documentos de la carpeta *Material*:
 - **Flash MX:** Versión de evaluación de Flash MX en castellano, restringida a un periodo de 30 días.
 - **Flash MX 2004:** Versión de evaluación de *Flash MX 2004* en inglés, restringida a un periodo de 30 días, con el fin de que pueda evaluar las características de la

nueva versión del programa. Para la plataforma Macintosh sólo podrá instalarse cuando el sistema operativo sea la versión 10.2.6 o superior.

- **Dreamweaver MX 2004:** Versión de evaluación de Dreamweaver MX 2004, en inglés, el editor HTML por excelencia, restringida también a un periodo de 30 días. Para la plataforma Macintosh sólo podrá instalarse cuando el sistema operativo sea la versión Mac OS 10.2.6 o superior.

- **Fireworks MX 2004:** Hemos incluido la versión de evaluación Fireworks MX 2004 en inglés, restringida a un periodo de 30 días, para que pueda evaluar las características de este nuevo programa. Con él podrá, entre otros, crear, diseñar, optimizar, y exportar gráficos interactivos en un sólo entorno centrado en la Web. Para la plataforma Macintosh sólo podrá instalarse cuando el sistema operativo sea la versión 10.2.6 o superior.

Todos los contenidos de este CD-ROM (programas, aplicaciones, utilidades, archivos, etc.) son shareware, freeware y demos, propiedad de las casas que junto a ellos se describen (en los archivos correspondientes), sujetos a sus condiciones de utilización, siendo necesario contactar con cada casa en cuestión si se produjera algún problema en su funcionamiento, licencia o temporalidad de utilización. En ningún caso **ANAYA MULTIMEDIA** se responsabiliza de su correcto funcionamiento.

Este CD-ROM es gratuito, y sólo se distribuye con el libro *Flash, PHP y MySQL. Contenidos dinámicos*, de la colección *Diseño y Creatividad*. Anaya Multimedia no se hace responsable de los programas ni de los daños que estos pudieran ocasionar por un uso indebido.

No obstante, para cualquier consulta puede dirigirse a nuestro soporte técnico, al teléfono (91) 320 90 36 o a nuestra dirección en Internet:
<http://www.AnayaMultimedia.es>

Índice alfabético

Operadores

+, 27
&, 51, 149, 183, 195
=, 51
#, 77
<?, 93, 153
<?php, 93
?>, 93, 153
//, 94
/*, 94
*/, 94

\$, 95
., 96, 97, 243
-, 102
\, 104, 106, 155
*, 126
[, 145
], 145
->, 145
?, 149, 251
\n, 155
“, 155
!, 157
..., 243

A

Acciones, Ventana de, 23-25, 30, 38
ActionScript, 16, 17, 23, 29, 34, 44, 280, 302, 304
Administrador, Panel de, 237, 247, 154, 262, 275, 280
AIFF, Documento, 47
AMF PHP, 301
ANSI, 51
Apache, Servidor Web, 18, 69, 71-73, 75, 76, 78, 79, 94, 120
Aplicaciones, Carpeta, 79
Apple, 75
AppServ, 70, 73-75, 94, 307
Array, 28, 29, 34, 57, 58, 97-100, 102, 104, 105, 108, 129, 137, 142-145, 159, 216, 239, 240, 245, 250, 258, 259, 282, 287, 288
Apuntador, 98-100, 239
Global, 258
Índice, 28, 97-99, 104, 105, 129, 142, 145, 259
array_push, 98
ASC, 128
asfunction, 287, 290
ASP, 15
attributes, 58
Auto_increment, 118

B

Base de datos, 15-17, 60, 67, 69, 72, 74, 81, 83, 84, 87, 102, 113, 119-121, 123, 124, 126, 128, 131, 132, 135-142, 145, 148, 152-154, 157, 164, 167, 170, 171, 174, 181, 182, 186, 187, 191, 192, 198, 202, 205, 207, 208, 211, 212, 221, 222,

224, 227, 231, 237, 247, 250, 253, 255, 256, 259, 262, 270, 271, 278, 293, 294
Biblioteca, 39, 41, 45, 47, 62-64, 66, 193, 196, 216
Bucle, 38-40, 98, 142, 145, 154, 160, 164, 166, 185, 187, 189, 192, 193, 196, 197, 221, 223, 243, 244, 257, 258, 260, 261

C

Caché, Memoria, 221, 291
Cadena, Tipo, 114, 117
Campo de texto, 43, 44, 48, 49, 53, 54, 61, 62, 64-67, 151, 152, 162, 164, 165, 178, 193, 194, 196, 197, 199, 214, 216, 238, 249, 251, 271, 272, 281, 284, 288, 290-292
Creación dinámica de, 43
Char, 117
Charset, 72
childNodes, 57, 58
Clases, 29, 40, 41
Clip de película, 29, 24-36, 39, 41, 42, 46, 47, 49, 50, 53, 54, 61, 62, 64-66, 181, 193, 194, 196-198, 216, 221, 234, 281, 284, 288, 291, 292
Creación dinámica de, 41, 61, 62, 288
Cold Fusion, 15
Columna, 113, 114, 118, 119, 127-129, 132, 141-146, 152, 154, 155, 157-159, 165, 167, 171, 172, 190, 211, 223, 250, 255, 259, 262, 293, 294
Compartir, 78
Computadora.de, 8
Condiciones, 36-39
config.inc.php, 74, 89
Consulta, 119, 128, 130, 137, 140, 141, 144, 146, 153, 154, 157-159, 162, 163,

170-172, 176, 182, 186, 188, 191-193,
199, 200, 203, 205, 206, 208, 211, 212,
218, 223, 224, 228, 231, 250, 259, 260,
262, 270, 293, 294
Ventana de, 120
Cookies, 95, 245, 250

D

Date, 221, 282
 getDate, 221
DateTime, 116
Datos, 113, 119
 Numérico, 26, 28
 Texto, 26, 28
 Booleano, 26, 28
Dec, 115
Decimal, 115
Default, 118
Delete, 33
DESC, 128
die, 136-138, 141
Disk Utility, 85
DMG, Documento, 85
Double, 115
Drawin' API, 41

E

each, 98, 99
echo, 155
ECMA-262, 17, 23
else, 37
empty, 158
enctype, 238
enum, 118
Estilos, Hoja de, 262, 265, 280
Estructuras condicionales, 37, 141, 224

exit, 167, 173, 176, 230
explode, 102
Expresión, 27, 32, 39, 64, 95, 96, 97, 106,
143, 145, 146, 196

F

false, 27, 48-50, 52, 57, 62, 106, 283
fclose, 108
Fecha, Tipo, 114, 115
Ficheros 106-108, 117, 228, 230, 239, 240,
261
 Apuntador, 107, 224
Fila, 119, 140, 142, 145, 152-156, 158,
159, 163, 175, 176, 191, 208, 212, 229,
243-245, 250, 260, 293
filesize, 107
fillInfo, 275
FLA, Documento, 16, 21, 23, 30, 31, 37,
39, 45, 47, 53, 60, 185, 190, 198, 200
Flash, 8, 16-18, 21-24, 26, 29-41, 43-45,
47, 49-55, 67, 97, 181, 182, 187, 190,
191, 193, 199, 205, 211, 213, 214, 221,
238, 254-256, 261, 268, 278, 282-284,
287, 288, 293, 297, 298, 301-305, 307
Flash MX, 17, 22-24, 39, 41, 43, 51, 54,
69, 181, 183, 206
Flash Remoting, 301, 304
float, 115
Flotante, Ventana, 272, 280, 287
fopen, 106, 107
for, 38-40, 52, 58, 62, 98, 185, 189, 193,
196, 221, 257, 258
Formato de texto, 43, 44
Formulario, 16, 121, 122, 148-152, 162-
167, 170, 171, 174-179, 222, 230, 237-
239, 246, 247, 249, 251, 254, 262, 265,
266, 271, 275, 278, 291, 297

Fotograma, 23, 26, 30, 33, 37, 38, 40-42,
44, 45, 50, 51, 56, 184, 189, 193, 194,
199, 207, 208, 213, 218, 232, 281, 282,
288, 291

Clave, 18

fputs, 108

fread, 107

FTP, 16, 69

Función, 31-36, 48, 56, 57, 60, 80, 94-96,
98-100, 102, 103, 106, 107, 109, 135-
137, 139-141, 146-149, 154, 155, 157-
159, 167, 173, 176, 187, 191, 194,
196-199, 202, 211, 214, 215, 220, 221,
223, 227-232, 239, 240, 243, 245, 250,
254, 256-259, 261, 266, 275, 282, 283
Global, 36

fwrite, 108

G

Generator, 16

Gestor *NetInfo*, 81, 82

GET, 95, 148-151, 155, 157, 158, 162,
163, 175, 195, 211, 212, 215, 217, 270,
287, 294, 295

GetImageSize, 240

GIF, Documento, 241

_global, 35, 36

H

htdocs, 94

HTML, 15, 16, 54, 55, 93, 96, 121, 143,
152-161, 164, 166, 170, 175, 176, 184,
214, 222, 243, 244, 256, 257, 265, 266
Documento, 93, 159, 198

Formato, 47, 53, 54, 61, 109, 110, 182,
193, 194, 241, 280

HTTP_POST_FILES, 239

http.conf, 78

I

if, 37, 141, 224, 241

implode, 102

include_path, 107

Index, 119

INSERT, 125, 126

Int, 114

Integer, 114

Instancias, 214

de clase, 29, 40

de clip de película, 39, 45, 62-65, 193,
194, 196-198, 216, 221, 234

Nombre de, 34-36, 62, 193, 281, 284

Internet, 15, 22, 69, 70, 307

Interpolación

de forma, 18

de movimiento, 18

IP, Dirección, 79

is_uploaded_file, 239

isset, 149, 157

Iteraciones, 38, 29, 98, 187, 193, 196, 243,
260

J

Java, 17

JavaScript, 17, 93, 251, 271, 272, 275, 280,
296

Jerarquías de clips, 34-36

JPEG, Documento, 22, 46, 47, 67, 237,
238, 240-242

JSP, 15

K

Key, 99, 101

L

Linux, 15

Límites, 129, 130

list, 146, 154, 293, 294

Lista 101, 142, 143, 146, 159, 239, 245

loadVariables, 181

loadVariablesNum, 181

LoadVars, 51, 52, 66, 181-186, 189, 190,
193-195, 198, 199, 201, 202, 205, 206,
208, 211, 213, 288

load, 52, 195, 202

onLoad, 48, 52, 184, 189, 194, 195, 199

send, 195

sendAndLoad, 195, 199, 202

localhost, 69, 72, 74

login, 246

logout, 246, 247

LongBlob, 118

LongText, 118

loop, 48

M

Macintosh, 15, 17, 47, 51, 69, 75, 77, 85,
307

mail, 109, 297

Matriz, 95

MAX_FILE_SIZE, 238

MediumBlob, 118

MediumInt, 114

Método, 34, 142, 152, 195, 211, 212, 239,
270, 293-295

MIME, 239

mktime, 106

Modo de edición

Experto, 24

Normal, 24

Módulos, 124, 246, 304

MovieClip, 41, 62, 196

_alpha, 41, 284

attachMovie, 62, 196

beginFill, 42

createEmptyMovieClip, 41

duplicateMovieClip, 41

endFill, 42

gotoAndPlay, 41

gotoAndStop, 41

lineStyle, 41

lineTo, 41

loadMovie, 47

moveTo, 41, 42

_name, 41

onRelease, 197

play, 41

Propiedades, 34

removeMovieClip, 288

_visible, 62, 67

MP3, Documento, 22, 47, 67

MS-DOS, 119

Multiparte, Formato, 238, 239, 266

my.ini Setup, 74

MySQL, 17, 18, 69-73, 75, 81, 83, 84, 87,
89, 113-115, 117, 119, 120, 122, 125,
126, 132, 135-138, 140, 141, 148, 154,
156, 160, 163, 166, 167, 173, 176, 181,
191, 230, 231, 254, 255, 259, 293, 294
Bases de datos, 69, 72, 81, 83, 84, 87,
89, 113, 139, 191, 208, 211, 224

Text, 117

MySQL Database Server, 84

mysql_connect, 137
mysql_error, 137, 141
mysql_fetch_array, 143, 146, 159, 250
mysql_close, 148
mysql_fetch_object, 145
mysql_fetch_row, 142, 146
mysql_free_result, 147
mysql_num_rows, 140
mysql_query, 140
mysql_select_db, 138

N

NaN, 196
.NET, 17
next, 98
nl2br, 294
Nodo, 55-58, 61-63, 65, 67, 206, 207, 210, 220, 221, 259-261, 290
 Atributos, 55, 61, 65, 67, 207, 214, 216, 220, 221, 260, 290
 Raíz, 57, 58, 61, 214, 216, 220, 259
nodeName, 57, 58
Not Null, 118
Null, 118
Numeric, 115
Numérico
 Tipo, 114, 241
 Valor, 27, 28, 52, 258

O

Objeto, 40, 48, 52, 54, 56, 57, 60-32, 65, 66, 142, 145, 182-185, 189, 190, 193-196, 198-202, 206, 207, 209-215, 217, 221, 232, 283, 284, 286, 288, 290
 Atributos, 40, 44, 194, 195, 199, 200
 Métodos, 40, 195, 196, 221, 283, 288

Open Source, 93, 113, 304
Operadores, 39
ORDER BY, 128
Orientada a Objetos, Programación, 17
OS X, 75, 81, 85

P

Paginación de resultados, 130
Parámetro, 24, 31, 32, 43, 49, 50, 52, 57, 62, 103, 104, 106-109, 146, 148, 152, 175, 195, 197, 199, 219, 221, 256-258, 266, 275, 283, 286-288, 297
_parent, 35
parseInt, 196
Parseo, 58
PC, 47, 51
PDF, Documento, 301
PHP, 15, 17, 18, 69, 70, 71, 75-78, 80, 93-98, 102, 104, 106, 109, 110, 135-138, 141, 148-150, 152-156, 158-160, 164-167, 170, 171, 173, 175, 176, 181, 182, 187, 189, 196, 198, 230, 238-241, 243
 date, 103, 105, 106, 266, 297
 Documento, 76, 79, 94, 102, 109, 136, 139, 141, 148-150, 152, 155, 162, 163, 170, 171, 176, 184-186, 189, 190, 192, 193, 195, 196, 203
 PHP Builder, 304
 PHP Net, 304
 PHP Object, 204
 PHP_SELF, 151, 164, 222, 251
 phpMyAdmin, 69-71, 74, 75, 87-90, 120, 231
 PHPSecurePages, 246
 PHPSESSID, 250
 pico, 77

Player, reproductor de *Flash*, 23
Plug-in, 17, 23
PNG, Documento, 241
POST, 95, 148, 149, 151, 152, 171, 172,
176, 190, 191, 193, 195, 199-203, 222
Preferencias del Sistema, 78, 82
Primary, 119
Prompt, 87

Q

Query, 135
QUERY_STRING, 251

R

readdir, 243
Registro, 87, 119, 124-132, 148, 152-155,
157, 162, 163, 167, 170, 174-177, 186,
187, 191, 192, 199, 211, 212, 217, 229,
237, 247, 250, 257, 259, 261-263, 265
Reproducir una película 30, 32-34, 36-40, 42-
46, 50, 52, 54, 58, 67, 185, 189, 207, 209
Reproductor de *Flash*, 23, 187
_root, 34, 36, 284
RTF, Documento, 80

S

Salida, Ventana de, 29-40, 52, 53, 57, 58,
185, 189, 207, 210, 287
Selection
 onKillFocus, 292
 onSetFocus, 291
Servicios, 78
Set, 118
Sesión, 95, 245-247, 249, 250, 254

_SESSION, 245, 250
session_destroy, 245, 250
session_start, 245
Shell, 76, 78, 87
Símbolos, Creación de, 23
SmallInt, 114
Socket, 17
Sound, 48-50
 loadSound, 49, 50
 onSoundComplete, 48
 setVolume, 49
 start, 50
 stop, 50
source, 272
String, 283, 287, 292
 split, 287
SQL, 113, 123, 124, 126, 127, 135, 141,
144, 146, 153, 154, 158, 162, 163, 171,
172, 176, 191, 192, 208, 223, 228, 231
 Documento, 122-124, 126
SQL2000, 17
SUBMIT, 151, 166, 171, 175, 177
sudo, 77
SWF, Documento, 22, 23, 31, 45-47

T

Tabla, 87, 113, 114, 117, 119-131, 140,
142, 143, 148, 152-154, 156-166, 170,
171, 176, 182, 184, 186-188, 190-193,
198, 199, 201, 207, 208, 211, 215, 217,
218, 221-224, 227, 229, 232, 243, 245
 Índice, 113, 119, 144
TCP/IP, Puerto, 113
Terminal, Aplicación, 76, 87
TextEdit, 79, 80, 89
TextField, 43, 284, 288
 htmlText, 48

maxscroll, 284, 288, 290
text, 43, 64
TextFormat, 43
this, 34, 194, 214
Time, 116, 240
TimeStamp, 116
TinyBlob, 118
TinyInt, 114
TinyText, 118
trace, 29, 31, 34
true, 27, 48, 50, 52, 56, 57, 61, 67, 157
TXT, Documento, 22, 51-54, 56, 60, 66

U

undefined, 34
Unicode, 51, 56, 187
Unique, 119
Universal, Hora, 221
Unix, 15, 75, 76, 78, 87
unset, 245
UPDATE, 171
URL, 54, 84, 130
utf_encode, 187
UTF-8, 187, 191, 208, 293
Utilidades, carpeta, 81

V

Value, 99, 101
Varchar, 117
Variable, 26-30, 32, 34-36, 38-40, 47, 48,
50-52, 54, 61, 63-66, 94-99, 101, 102,
105, 107, 136, 138, 140-142, 145-152,
154, 155, 157-159, 162-164, 167, 170-
172, 175, 176, 183-196, 199, 202, 203,
207, 208, 211, 212, 214-216, 221-224
Locales, 32, 33, 95, 96

Globales, 32, 36, 95, 96, 140
Superglobales, 95, 96
Vinculación, Nombre de, 62, 63, 193

W

WAV, Documento, 47
where 128, 132, 167, 176
while 38-40, 142, 145, 154, 160, 164, 166,
187, 192, 223, 244, 261
_width, 29
with, 42
Windows, 15, 17, 69, 73, 82, 106, 307
WinMySQLAdmin, 73
WYSIWYG, 159

X

_x, 41, 42, 64
XML, 17, 54, 61, 205-209, 211, 217, 221,
222, 224, 228, 231
Árbol, 57, 58, 62, 206, 208, 212, 216, 217,
220, 227, 234, 259
Documento, 23, 31, 54-57, 60, 61, 67,
206, 221-224, 229, 231, 237, 259-261
Objeto, 54, 56, 60, 61, 62, 65, 182, 206,
207, 209, 210, 214, 215, 217, 232, 290
firstChild, 57
ignoreWhite, 56, 61, 210, 214, 290
onLoad, 56, 57, 60, 67, 207, 215, 290
sendAndLoad, 215
XReal, 115

Y

_y, 41
Year, 116