

Introducción

Utilizaremos este primer capítulo para explicar los conceptos básicos referentes a cada lenguaje a fin de poder encarar sin problemas el desarrollo de los proyectos de este libro.

Conceptos generales	14
PHP y MySql	14
XML	24
ActionScript y los datos externos	32
Objeto LoadVars	33
Objeto XML	42
Conclusión	54
Resumen	55
Actividades	56

CONCEPTOS GENERALES

Este primer capítulo es de carácter teórico y no tiene como finalidad enseñar a programar en los lenguajes que se desarrollan a lo largo del libro.

En los ejemplos, utilizaremos **bases de datos**, **PHP** del lado del servidor, **Flash** del lado del cliente y **XML** como nexo entre ambos.

Lo que nosotros haremos será tratar los **conceptos necesarios** para entender con claridad los códigos que emplearemos a partir del próximo capítulo; de este modo, no explicaremos conceptos básicos de ninguno de estos lenguajes (damos por entendido que los lectores de este libro tienen ideas básicas de programación o programan en alguno de los lenguajes que trataremos).

Si recién está dando sus primeros pasos en la materia, quizás sea conveniente que tenga cerca algún otro material de consulta por si se hace necesario. La finalidad de este libro es entender la interacción de estos lenguajes, y no su funcionamiento de manera autónoma. En este capítulo, repasaremos:

- **PHP y MySQL**: explicaremos de qué manera almacenamos valores en una base de datos utilizando código **PHP**.
- **XML**: conceptos básicos de este metalenguaje basado en el uso de etiquetas.
- **PHP, MySQL y XML**: veremos de qué forma podemos crear con **PHP** estructuras **XML** utilizando para dichas estructuras los valores almacenados en una base de datos o cualquier otra fuente.
- **ActionScript**: veremos el objeto **LoadVars()** empleado para enviar y recibir variables desde **Flash** y hacia **PHP**, y desde **PHP** hacia **Flash**. También veremos el objeto **XML** para ver de qué manera interpretamos las estructuras **XML** que crearemos con **PHP**.

Hechas estas salvedades, damos inicio al repaso teórico. De más está decir que si ya manejamos alguno de estos lenguajes, o poseemos experiencia previa, tranquilamente puede obviar las explicaciones referentes a dicho lenguaje y continuar con los conceptos que desconozcamos, o bien comenzar a ver el segundo capítulo, donde ponemos en juego estos lenguajes simultáneamente.

PHP y MySQL

¿Qué es una base de datos?

Una base de datos es una aplicación informática cuya finalidad es el almacenamiento de información relacionada.

¿Cómo se estructura una base de datos?

Las bases de datos contienen, en un principio, **tablas**; las **tablas** cumplen la función

de contener **campos**; y cada **campo** contiene registros. Veremos esto de forma clara al crear una en el próximo ejemplo.

¿Qué es SQL?

SQL es un lenguaje de consulta estructurado (*Structured Query Language*) que nos permite realizar diversos tipos de operaciones sobre las bases de datos.

¿Qué es phpMyAdmin?

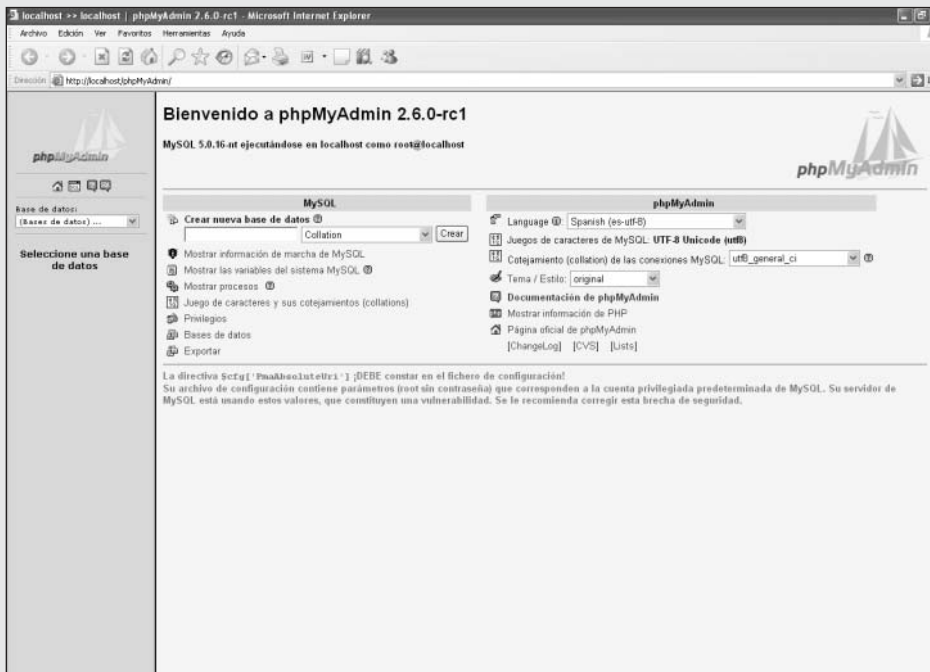
Es un proyecto de código abierto, programado en **PHP**, cuya finalidad es la administración eficiente de bases de datos **MySQL** a través de una interfaz web.

Con phpMyAdmin podemos crear y eliminar bases de datos, crear y eliminar tablas, editar campos (cambiar los tipos de datos, longitud, incremento, etc.), ejecutar cualquier sentencia SQL y administrar privilegios (mediante Roles y Usuarios), entre muchas cosas más. Utilizaremos **phpMyAdmin** para crear las bases de datos que usaremos a lo largo de este libro.

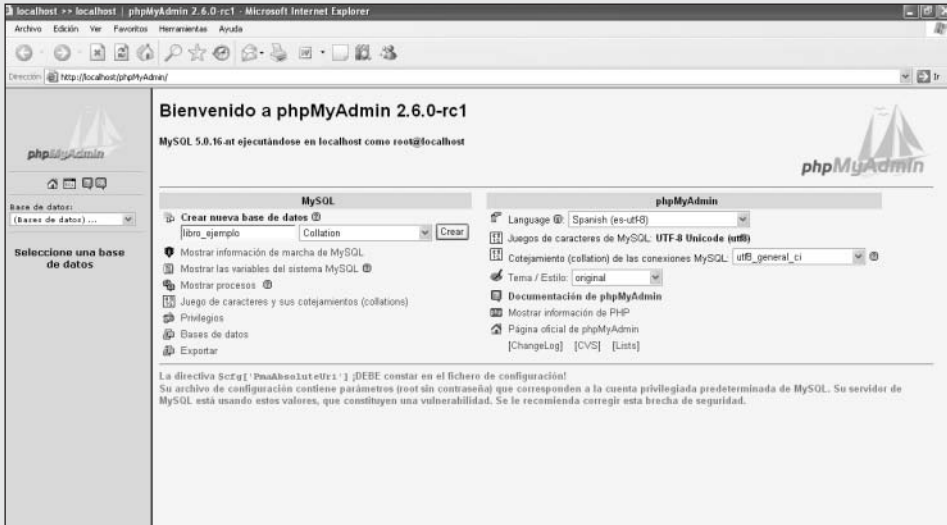
■ Creación de una base de datos

PASO A PASO

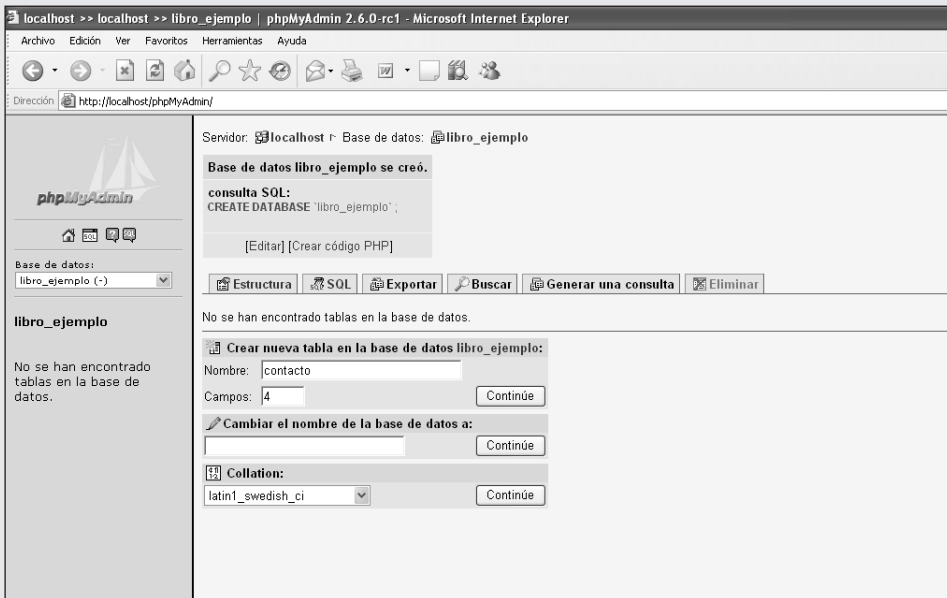
- 1 Ingrese a **phpMyAdmin** escribiendo en su explorador **http://localhost/phpMyAdmin**.



2 Introduzca el nombre de la base de datos que desea crear donde se le indica y presione Crear. En nuestro caso, la llamaremos **libro_ejemplo**.

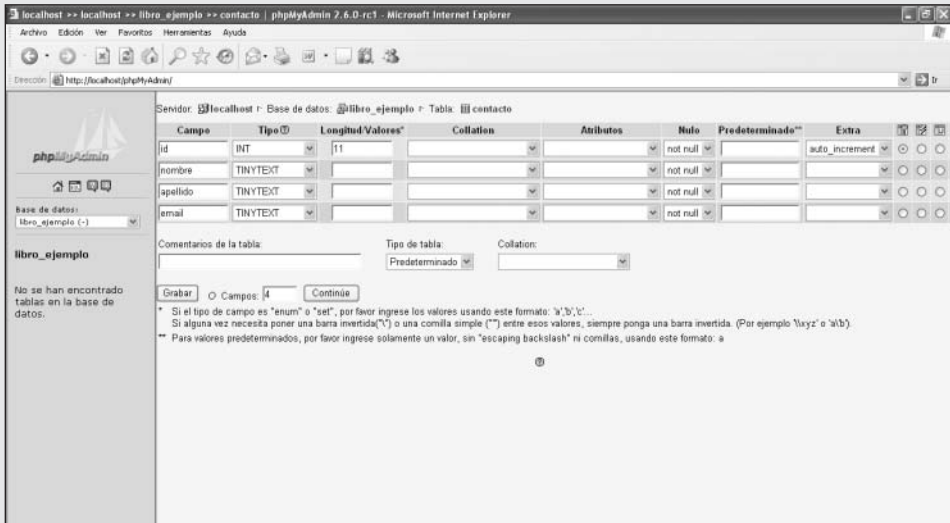


3 Una vez creada nuestra base de datos, debemos crear tablas dentro de ella; para esto escriba su nombre e indique la cantidad de campos que tendrá, y presione continuar. En nuestro caso, llamaremos Contacto a la tabla, la cual tendrá 4 campos.



4

Nombre a los campos de la siguiente manera: **ID, nombre, apellido, e-mail**.



Con respecto a los campos que vamos a crear, hay algunas particularidades para tener en cuenta: en la columna que lleva por nombre **Campo**, debemos escribir el nombre del campo. Al lado de la columna de los campos, tenemos la de los **tipos**. Aquí debemos indicar el tipo de datos que contendrá cada columna, pero debemos tener en cuenta que existen diversos tipos de datos y debemos ser cuidadosos con esto:

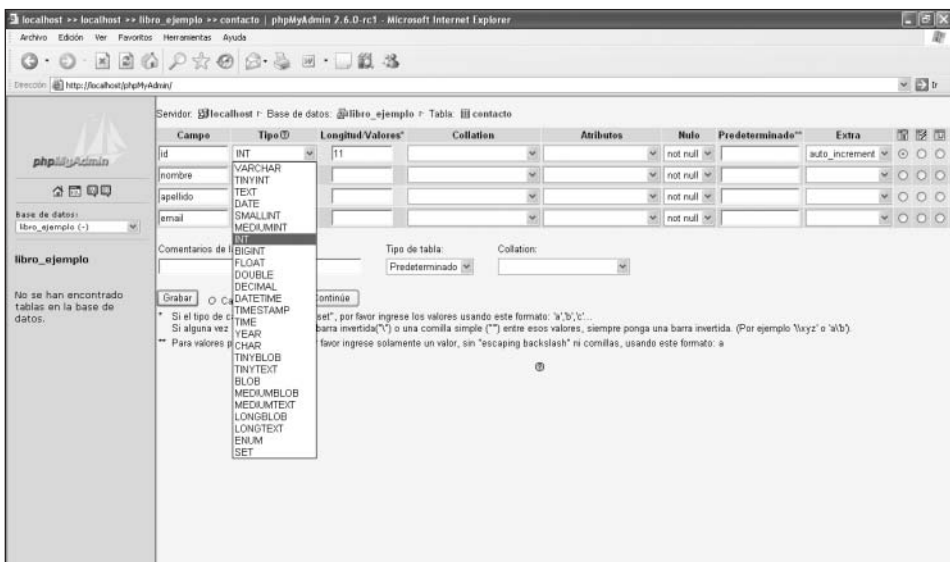


Figura 1. Tipos de datos disponibles para los campos en MySQL.

Tipos de datos:

En este apartado, veremos qué tipos de datos están disponibles, y qué valores soporta cada uno de ellos mediante una tabla comparativa.

TIPO DE DATO	DESCRIPCIÓN
TinyInt	Nos permite almacenar un número entero con signo o sin él. En caso de que el valor lleve signo, el rango de valores válidos va de -128 a 127. Sin signo, el rango de valores va de 0 a 255.
SmallInt	Nos permite almacenar un número entero con signo o sin él. Con signo, va de -32.768 a 32.767. Sin signo, va de 0 a 65.535.
MediumInt	Número entero. Con signo, el rango de valores va de -8.388.608 a 8.388.607. Sin signo, el rango va de 0 a 16.777.215.
Int	Número entero. Con signo, el rango de valores va de -2.147.483.648 a 2.147.483.647. Sin signo, el rango va desde 0 a 4.294.967.295.
BigInt	Número entero. Con signo, el rango de valores va de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo, el rango va de 0 a 18.446.744.073.709.551.615.
Float	Número flotante. Los valores van de -3.402823466E+38 a -1.175494351E-38, 0 y de 1.175494351E-38 a 3.402823466E+38.

Tabla 1. Tipos de datos numéricos.

TIPO DE DATO	DESCRIPCIÓN
Char	La cadena podrá contener de 0 a 255 caracteres.
VarChar	La cadena podrá contener de 0 a 255 caracteres.
TinyText	Permite una longitud máxima de 255 caracteres.
Text	Permite un máximo de 65.535 caracteres.
MediumText	Permite un máximo de 16.777.215 caracteres.
LongText	Permite un máximo de 4.294.967.295 caracteres.

Tabla 2. Tipos de datos de cadenas.

TIPO DE DATO	DESCRIPCIÓN
Date	El rango de valores va del 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es YYYY-MM-DD.
DateTime	Nos permite almacenar fecha y hora simultáneamente. El formato de almacenamiento es YYYY-MM-DD HH:MM:SS.
Time	Almacena una hora. El formato de almacenamiento es 'HH:MM:SS'
Year	Almacena un año. El rango de valores permitidos va del año 1901 al año 2155.

Tabla 3. Tipos de datos de fecha.

Una vez que definimos el tipo de dato, podemos definir otros comportamientos, de los cuales, si bien son opcionales, más de uno nos será necesario:

COMPORTAMIENTO	DESCRIPCIÓN
Null_NotNull	En caso de que el valor sea Null, se permitirá que se inserten valores vacíos en nuestra base de datos. Por defecto, esta opción viene seteada en Null.
Predeterminado	Define el valor por defecto con el cual se va a crear el nuevo registro, siempre y cuando no se le asigne ninguno.
Extra - Auto Increment	Almacenará automáticamente el valor numérico más alto de toda la columna. El incremento se hará de uno en uno.
Primary Key	Nos permite definir una columna como la "principal" de nuestra tabla. Veremos que haremos uso de esta opción a lo largo de todos los ejemplos, ya que nos permitirá realizar la búsqueda de los registros.

Tabla 4. Comportamientos.

Bien, hasta aquí vimos cómo creamos una base de datos; nos resta ver de qué manera se introducen datos en ella y cómo se extraen:

Utilizamos **Insert Into** para insertar datos en una tabla de nuestra base de datos:

```
INSERT INTO `tabla` (`campo`, `campo`) VALUES (`valor`, `valor`);
```

Utilizamos **Select** para seleccionar datos de una fila:

```
SELECT * FROM `tabla` WHERE id='fila';
```

Utilizamos **Delete** para borrar una fila:

```
DELETE FROM `tabla` WHERE id='el número de la fila';
```

Utilizamos **Like** para buscar datos dentro de una tabla:

```
SELECT campo FROM tabla WHERE campo LIKE LIKE '%"lo que queremos buscar"%;
```

III AUTO INCREMENT Y PRIMARY KEY

Utilizaremos estos comportamientos a lo largo del libro ya que con ellos podremos llevar un conteo ordenado y organizado de los registros en nuestra base de datos. Generalmente llamaremos **id** al campo que represente este valor autoincrementable.

Si bien este material teórico le puede servir para despejar dudas, seguramente entenderá de un mejor modo el funcionamiento de una base de datos con un ejemplo práctico. Crearemos una base de datos y, desde **PHP**, almacenaremos registros en ella.

Ejemplo práctico:

La base de datos: **libro_ejemplo**

Para este ejemplo, utilizaremos la base de datos que creamos anteriormente. Ahora veremos el archivo **nuevoRegistro.php**:

```

1 <?php
2     require("conect.php");
3     $phpNombre = "Mariano";
4     $phpApellido = "Makeeee";
5     $phpEmail = "mimail@miservidor.com.ar";
6     $consulta = mysql_query("INSERT INTO contactos (`id`, `nombre`, `apellido`, `email`)
7     VALUES ('', '$phpNombre', '$phpApellido', '$phpEmail');");
8     if($guardar == mysql_query("$consulta"))
9     {
10        echo "se agregaron nuevos datos a la BD";
11    }else{
12        echo "error";
13    }
14 ?>
15
16

```

Figura 1. El archivo *nuevoRegistro.php* se encargará de almacenar tres datos. Luego, veremos que dichas variables se enviarán desde formularios **HTML** o desde **Flash**.

```

<?php

    require("conect.php");
    $phpNombre = "Mariano";
    $phpApellido = "Makeeee";
    $phpEmail = "mimail@miservidor.com.ar";

```

WWW.MYSQL.COM/

Sitio oficial de **MySQL**. Encontrará muchísima información respecto a este tipo de base de datos. Puede ser de gran utilidad contar con este sitio como material de consulta, para cuando desee realizar proyectos de mayor envergadura.

```

        $consulta = mysql_query("INSERT INTO contactos (`id`, `nombre`,
`apellido`, `email`) VALUES ('', '$phpNombre', '$phpApellido', '$phpEmail');");
        if($guardar == mysql_query("$consulta"))
        {
            echo "se agregaron nuevos datos a la BD";
        }else{
            echo "error";
        }
    }
?>

```

Utilizamos las etiquetas `<?php` y `?>` para indicar que todo código que se encuentre entre ellas pertenece a **PHP**.

Lo primero que encontramos es la siguiente línea:

```
require("conect.php");
```

La función **require()** nos sirve para incluir archivos, funciones o fracciones de código; generalmente se emplea para agregar cabeceras o pies de página. Nosotros la usaremos para llamar al archivo **conect.php** y así conectarnos a la base de datos.

Un dato importante para tener en cuenta es que, a lo largo de todos los ejemplos del libro, utilizaremos el archivo **conect.php**; simplemente modificaremos el nombre de la base de datos a la cual deseamos conectarnos:

conect.php

```

<?
    $Servidor = localhost;
    $Usuario = "root";
    $Password = "";
    $BaseDeDatos = "libro_ejemplo";
    $conexion=mysql_connect($Servidor,$Usuario,$Password) or die("Error: El
servidor no puede conectarse con la base de datos");
    $descriptor=mysql_select_db($BaseDeDatos,$conexion);
?>

```

En primer lugar, creamos cuatro variables: **\$Servidor**, **\$Usuario**, **\$Password** y **\$BaseDeDatos**. Luego definimos sus valores.

Para averiguarlos, ingrese a **phpMyAdmin** y luego al link **privilegios**; ahí encontrará la información relativa al nombre del servidor, al usuario y a la contraseña.



```

1 <?
2 $Servidor = localhost;
3 $Usuario = "root";
4 $Password = "";
5 $BaseDeDatos = "libro_ejemplo2";
6 $conexion=mysql_connect($Servidor,$Usuario,$Password) or die("Error: El servidor no puede conectar con la base de datos");
7 $descriptor=mysql_select_db($BaseDeDatos,$conexion);
8 ?>

```

Figura 2. El archivo *connect.php* se utiliza a lo largo de todos los proyectos del libro; sólo debemos modificar el valor de la variable *BaseDeDatos*.

Luego declaramos la variable **\$conexion** y utilizamos la función **mysql_connect** con la cual abrimos una conexión a un servidor **MySQL**; los parámetros que debemos pasarle a dicha función por valor son:

mysql_connect (el servidor, el usuario, la contraseña);

Por lo tanto, para conectarnos, utilizamos las tres variables creadas anteriormente.

```

$conexion=mysql_connect($Servidor,$Usuario,$Password) or die("Error: El servidor
no puede conectarse con la base de datos");

```

III CONEXIÓN A LA BASE DE DATOS

La ventaja de incluir la conexión a la base de datos mediante la función **require()** radica en que podemos utilizar siempre el mismo archivo. De este modo, simplemente tenemos que abrir el archivo **connect.php** y modificar el nombre de la base de datos a la cual deseamos acceder.

Lo que hicimos hasta el momento fue conectarnos con la base de datos; ahora, nos queda seleccionar la base de datos; para esto utilizamos la función `mysql_select_db`. Los parámetros que debemos asignarle son el nombre de la base de datos, para lo cual vamos a utilizar la variable anteriormente creada (`$BaseDeDatos`), y luego el identificador de enlace; para esto utilizamos la variable `$conexion`. Esta función nos devuelve **True** en caso de que la selección de la base de datos se lleve a cabo correctamente, o **False** en caso contrario.

Una vez visto cómo establecemos la conexión a la base de datos y seleccionamos la DB (*Database* o Base de Datos) que vamos a utilizar, continuamos con **nuevoRegistro.php**

```
$phpNombre = "Mariano";
$phpApellido = "Makeeee";
$phpEmail = "mimail@miservidor.com.ar";
$consulta = mysql_query("INSERT INTO contactos (`id`, `nombre`, `apellido`,
`email`) VALUES ('', '$phpNombre', '$phpApellido', '$phpEmail');");
```

En primer lugar, creamos las tres variables que vamos a insertar en la base de datos; las llamamos `$phpNombre`, `$phpApellido` y `$phpEmail`. Una vez definidos sus valores, utilizamos la función `mysql_query()` para enviar una consulta a **MySQL**. Anteriormente explicamos de qué modo insertamos datos dentro de una tabla; utilizamos **Insert Into**, seguido del nombre de la **tabla**, y luego los **campos** de dicha tabla en los cuales vamos a introducir los valores. A continuación, dentro de **Values**, indicamos cuáles serán dichos valores (observe que cada una de las variables de **Values** se corresponde con los campos de la tabla en cuestión).

```
$consulta = mysql_query("INSERT INTO nombre tabla (`campo`, `campo`) VALUES
(`valor`, `valor`);");
```

`mysql_query` devolverá **true** en caso de que el ingreso de datos se haga de forma



CONTENIDOS DE LAS BASES DE DATOS

En este caso, como se trata de un sencillo ejemplo, simplemente declaramos tres variables en **PHP** y luego las almacenamos. Lógicamente, esto no tiene demasiado sentido a la hora de desarrollar un proyecto, pero para esto mismo veremos luego de qué manera enviar dichas variables desde formularios, ya sea desde **HTML** o desde **Flash**, por medio del objeto **LoadVars()**.

correcta, y **false** en caso contrario. Por esto mismo, utilizamos el condicional **if()** a fin de indicar si el ingreso de datos se realizó de manera satisfactoria.

```

if($guardar == mysql_query("$consulta"))
{
    echo "se agregaron nuevos datos a la BD";
}else{
    echo "error";
}
    
```

Si ejecutamos este código dentro del servidor, veremos que introduciremos los nuevos valores a la base de datos.

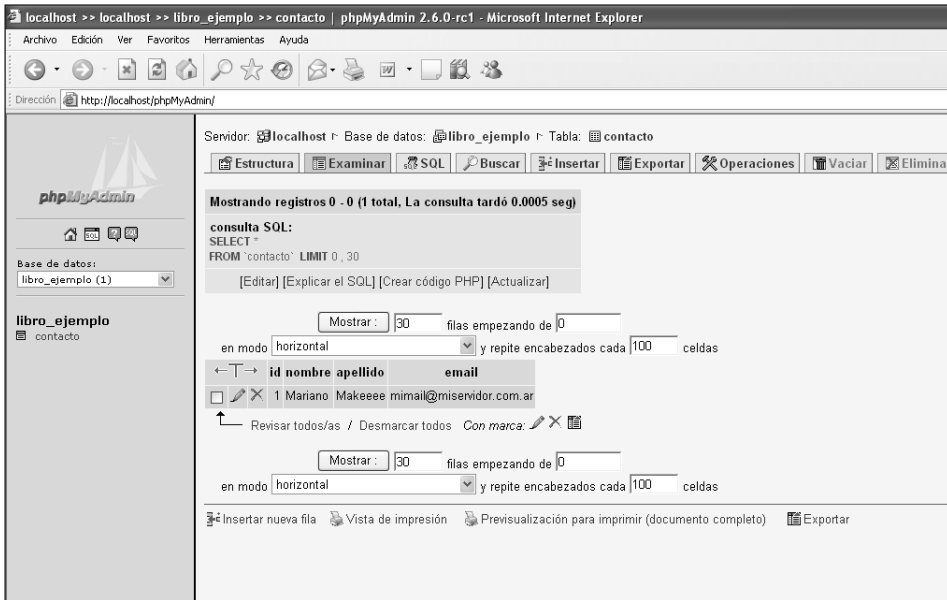


Figura 3. Una vez que ejecutamos el archivo *nuevoRegistro.php*, se almacenan en la base de datos las variables que creamos.

XML

A nuestro criterio y teniendo en cuenta el uso que le daremos, **XML** (*eXtensible Markup Language*) es un estándar para el intercambio de información estructurada entre diferentes plataformas y tiene la inmensa ventaja de permitir la compatibilidad entre distintos tipos de sistemas, gracias a lo cual se puede compartir información de forma segura y, por sobre todas las cosas, de un modo sencillo.

No profundizaremos ni teorizaremos acerca de **XML** dado que existen innumerables contenidos en la Red; nuestra prioridad es explicar la sintaxis que utiliza, la cual, como veremos, es verdaderamente sencilla.

Sintaxis:

Podemos dividir la estructura en dos partes principales: la cabecera y el contenido del archivo, el cual se segmenta en etiquetas.

Cabecera:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Código dentro de **TEXTO** es comando.

Es imprescindible recordar escribirla a fin de que el documento se interprete correctamente como una estructura **XML**.

Etiquetas:

Los documentos se basan en la apertura y el cierre de etiquetas predefinidas por el usuario; esto quiere decir, sencillamente, que a diferencia de **HTML**, donde tenemos etiquetas predefinidas, aquí podemos llamarlas del modo que deseemos e incluirles los valores que necesitemos.

La estructura que presenta un documento **XML** es jerárquica; esto implica la existencia de nodos padres, nodos hijos, nietos, y podríamos alargar la familia hasta donde lo deseáramos:

```
<padre>
  <hijo></hijo>
</padre>
```

Como podemos ver, dentro de las etiquetas de apertura y de cierre, podemos incluir otras etiquetas de apertura y de cierre. Para cerrar una etiqueta, lo hacemos del mismo modo que en **HTML**: interponemos la barra (/) al nombre de la etiqueta.



WWW.PHP.NET/MANUAL/ES/

Manual en español del lenguaje **PHP**. Se encuentra dentro de la página oficial de dicho lenguaje. Nos puede resultar útil para realizar diversos tipos de consultas y el funcionamiento de distintas funciones, así como las declaraciones de variables y los tipos de datos disponibles.

El valor de las etiquetas se define en su interior:

```
<padre>
  <hijo>Éste es el valor de la etiqueta hijo!</hijo>
</padre>
```

Otro dato importante para tener en cuenta es que las etiquetas pueden contar con atributos; podemos incluir tantos atributos como queramos:

```
<padre atributo1 = "esto es un atributo" atributo2 = "otro atributo">
  <hijo>Éste es el valor del nodo</hijo>
</padre>
```

Las estructuras **XML** son sencillas; su importancia no sólo radica en la apertura y el cierre de las etiquetas, sino en nuestra capacidad de organizar de forma lógica y ordenada la información que deseamos mostrar. Lo que queremos decir con esto es que existen diversos modos de dar a entender la información que deseamos por medio de **XML**, pero la importancia de esto radica en hacerlo de un modo ordenado y prolijo. Supongamos que queremos generar un **XML** que contenga datos referentes a un catálogo discográfico:

```
<discos>
  <disco nombre = "el nombre" año = "2007"
    <comentarios> Éstos son los comentarios referentes al disco. Lo que queremos
    mostrar por medio de este sencillo ejemplo es que, en más de una oportunidad,
    tenemos que pensar de qué manera vamos a mostrar la información a fin de
    hacerlo de la forma más ordenada. Imagine que ponemos estos comentarios como
    un atributo del nodo disco; si bien podríamos acceder a la información, no
    parece ser un modo organizado de mostrar los contenidos </comentarios>
  </disco>
  <disco nombre = "otro disco" año = "2006"
    <comentarios> Éstos son los comentarios del segundo disco </comentarios>
  </disco>
</discos>
```

Si bien existe innumerable cantidad de teoría respecto a **XML**, el único modo de entender completamente el funcionamiento de este metalenguaje, es practicar. Con el correr del tiempo, iremos creando estructuras **XML**, desde las más sencillas hasta las

que requieren cierta complejidad en su organización. Esta organización depende en gran medida de nosotros, y la práctica nos dará el modo de saber llevarla a cabo.

Ejemplo:

Un dato para tener en cuenta es que podemos generar las estructuras **XML** de dos maneras; podemos escribirlas en forma manual, o bien podemos recorrer una base de datos con **PHP** e ir extrayendo los valores de esa base, y mostrar dicha información estructurada en un documento **XML**.

A lo largo del libro, crearemos nuestras estructuras con **PHP** (veremos la innumerable cantidad de ventajas que esto nos da).

En el ejemplo anterior, lo que hicimos fue incluir valores en la base de datos. Ahora extraeremos esos valores con **PHP** y generaremos una estructura **XML** con los contenidos antes mencionados.

La base de datos **libro_ejemplo2**:

Nuestra base de datos cuenta con una única tabla, que llamamos **personitas**. En su interior, la **tabla** en cuestión contiene cinco **campos**:

id

nombre

apellido

email

comentarios

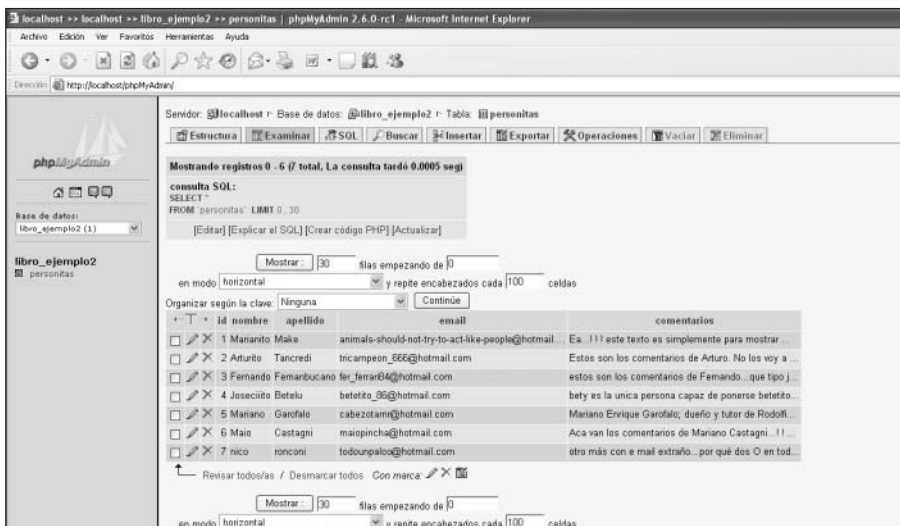


Figura 4. Presionando **Examinar** podemos ver los contenidos dentro de una tabla. **PHP** recogerá estos datos y los organizará en una estructura **XML**.

Nos interesa poder recorrer la base de datos e imprimir los valores y conformar así una estructura **XML**. Para esto utilizaremos el archivo **armarEstructura.php**.



```

1 <?php header("Content-type: text/xml\n\n"); ?>
2 <?php echo ('<?xml version="1.0" encoding="ISO-8859-1"?>' . "\n"); ?>
3 <?php
4     require("conect.php");
5     echo "<personas>\n";
6     $consulta = mysql_query("SELECT * FROM `personitas` ORDER BY id ASC");
7     while($resultado = mysql_fetch_array($consulta))
8     {
9         echo "<contacto>\n";
10        echo "<personal nombre = \"\$resultado[nombre]\" apellido = \"\$resultado[apellido]\"/>\n";
11        echo "<e-mail>\$resultado[email]</e-mail>\n";
12        echo "<comentarios>\$resultado[comentarios]</comentarios>\n";
13        echo "</contacto>\n";
14    }
15    echo "</personas>\n";
16
17 ?>

```

Figura 5. El archivo **armarEstructura.php** se encargará de crear una estructura XML.

armarEstructura.php

```

<?php header("Content-type: text/xml\n\n"); ?>
<?php echo ('<?xml version="1.0" encoding="ISO-8859-1"?>' . "\n"); ?>
<?php
    require("conect.php");
    echo "<personas>\n";
    $consulta = mysql_query("SELECT * FROM `personitas` ORDER BY id ASC");
    while($resultado = mysql_fetch_array($consulta))
    {
        echo "<contacto>\n";

```

III VENTAJAS DE TRABAJAR CON XML

Antes de que Flash incluyera el objeto **XML()**, se utilizaba **LoadVariables** para la carga de datos externos. La inclusión del objeto **XML()** simplificó la carga de estos datos, ya que permitió una organización mucho más prolija y organizada. La compatibilidad de **PHP** para crear dichas estructuras y de **Flash** para interpretarlas permite una inmensa gama de posibilidades para desarrollar.

```

    echo "<personal nombre = \"\$resultado[nombre]\" apellido = \"\$resultado
[apellido]\"/>\n";
    echo "<e-mail>\$resultado[email]</e-mail>\n";
    echo "<comentarios>\$resultado[comentarios]</comentarios>\n";
    echo "</contacto>\n";
}
echo "</personas>\n";
?>

```

Iremos explicando cada línea del siguiente código:

```
<?php echo ('<?xml version="1.0" encoding="ISO-8859-1"?>' . "\n"); ?>
```

Al explicar **XML**, dijimos que necesitábamos imprimir el encabezado del documento a fin de que se lo reconociera como una estructura **XML**. Para imprimir dicha línea, lo hacemos por medio de **echo**.

Utilizamos `\n` para generar un salto de línea. Vamos a hacer uso constantemente de saltos de línea ya que, para crear correctamente las estructuras **XML**, necesitamos de ellos. Una vez que imprimimos el encabezado, llamamos al archivo **conect.php**

```
require("conect.php");
```

Recordemos que utilizamos ese archivo para conectarnos a la base de datos (ya explicamos anteriormente su funcionamiento).

Una vez conectados, imprimimos la primera línea de la estructura **XML** con **echo** y luego forzamos un salto de línea:

```
echo "<personas>\n";
```



LENGUAJE DEL LADO DEL SERVIDOR: ¿PHP O ASP?

Decidimos utilizar **PHP** dado que presenta importantes ventajas: **PHP** es un lenguaje **open source** (de código abierto), podemos utilizarlo en distintas plataformas, tiene soporte para diversos servidores, el acceso a bases de datos es verdaderamente sencillo y es seguro, entre otras muchas cualidades. De todos modos, la interpretación de esas estructuras desde **Flash** es exactamente igual.

Generamos la variable **\$consulta**, la cual contendrá el **Query** (explicamos anteriormente que utilizamos **Select From** para extraer valores de una determinada tabla). En nuestro caso, definimos la tabla **personitas** y ordenamos los resultados de la consulta de forma ascendente teniendo en cuenta el campo **id** de la tabla.

```
$consulta = mysql_query("SELECT * FROM `personitas` ORDER BY id ASC");
```

Aquí encontramos un nuevo concepto; **mysql_fetch_array**. Lo que hacemos con **mysql_fetch_array** es extraer la fila de un resultado como una matriz asociativa. Es decir, nos devuelve una matriz ordenada que contiene todos los datos de una fila; en nuestro caso, se trata del **id**, el **nombre**, el **apellido**, el **e-mail** y un **comentario** de una persona.

Incluimos la sentencia dentro del bucle **while**:

```
while($resultado = mysql_fetch_array($consulta))  
{
```

Es importante que prestemos atención a lo que hacemos dentro del ciclo de repetición. En primer lugar, imprimimos la apertura de la etiqueta **contacto** y generamos un salto de línea:

```
echo "<contacto>\n";
```

Luego imprimimos el nombre y el apellido de la persona; recordemos que dichos datos fueron incluidos dentro de un **array**, el cual asociamos a la variable **\$resultado**. Por este mismo motivo, ingresamos a los datos del siguiente modo:

```
echo "<personal nombre = \"\$resultado[nombre]\" apellido = \"\$resultado  
[apellido]\"/>\n";
```

Extraemos el campo **nombre** del array **\$resultado** y luego el campo **apellido**. Lo mismo hacemos con el e-mail:

```
echo "<e-mail>$resultado[email]</e-mail>\n";
```

Por último, extraemos los comentarios:

```
"<comentarios>$resultado[comentarios]</comentarios>\n";
```

Una vez extraídos todos los datos del array **\$resultado**, debemos cerrar la etiqueta **contacto**. Recordemos que lo hacemos interponiendo la barra (/) en la etiqueta.

```
echo "</contacto>\n";
```

Una vez fuera del bucle **while**, cerramos la etiqueta principal de nuestra estructura y finalizamos nuestro código **PHP**.

```
}
echo "</personas>\n";
?>
```

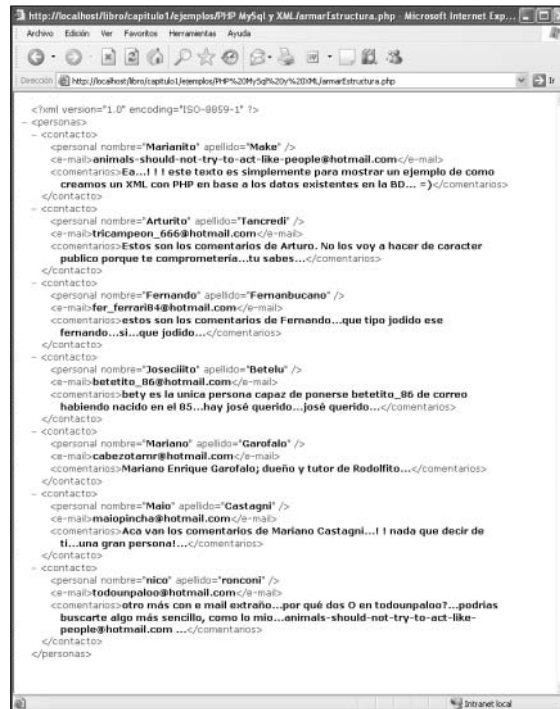


Figura 6. Si ejecutamos el archivo *armarEstructura.php* en el servidor, se imprime dicha estructura XML.

Una particularidad para tener en cuenta, posteriormente, al generar estructuras **XML** con **PHP** es el uso de las comillas.

```
echo "<personal nombre = \"\$resultado[nombre]\" apellido = \"\$resultado
[apellido]\"/>\n";
```

Si analizamos dichas líneas, vemos que por un lado **PHP** necesita del uso de comillas para imprimir las líneas, y **XML** para imprimir los **atributos**. Cuando queramos incluir contenidos con comillas, deberemos interponer la barra invertida (\) y las comillas (") al comienzo y al final de los contenidos que se van a imprimir.

ActionScript y los datos externos

Con el correr del tiempo, **Flash** pasó de ser un simple programa de desarrollo de animaciones vectoriales a ser una tecnología de producción multimedia que abarca distintos campos y le da la justa importancia a la posibilidad de trabajar con datos externos, ya sean imágenes, sonidos o videos, entre otros alcances. A su vez, el objeto **LoadVars** y las estructuras **XML** enriquecieron nuestras posibilidades, ya que gracias a ellos podemos enviar y recibir datos desde nuestras películas y hacia ellas. Del uso de estos recursos, nace este libro; a lo largo de los distintos proyectos, los iremos utilizando según nuestra conveniencia, por lo cual haremos un repaso teórico acompañado de ejemplos para ir entendiendo su funcionamiento.

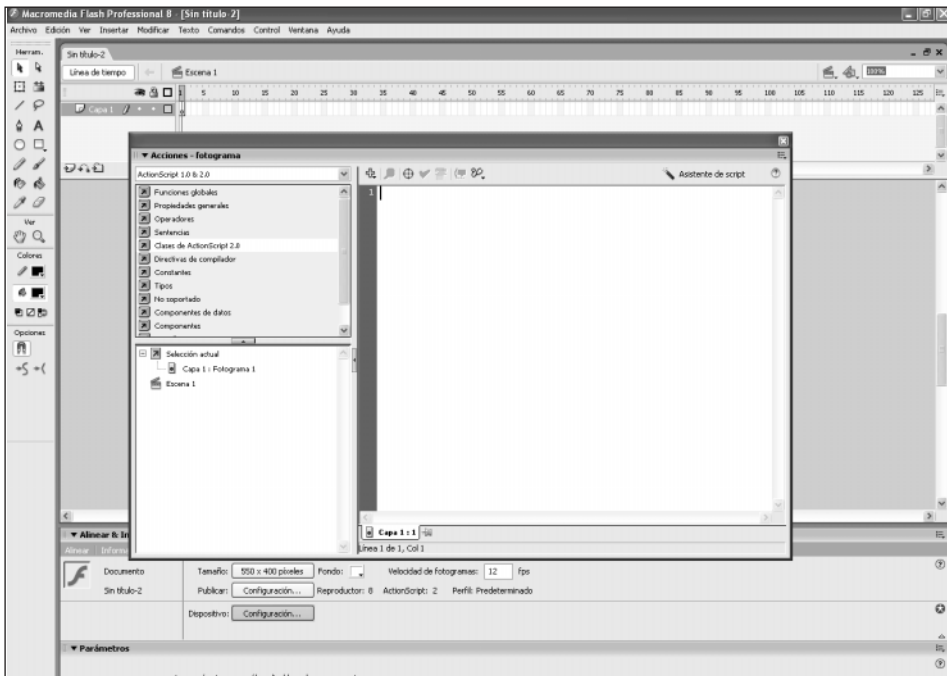


Figura 7. Desarrollaremos nuestros códigos en el Panel de Acciones. Puede ingresar a ese panel presionando **F9**, que es la tecla de acceso rápido.

Objeto LoadVars

El objeto **LoadVars** se encuentra disponible a partir de la versión **MX** de **Flash**, y su finalidad es el envío y la recepción de datos con archivos externos. Si bien en nuestros proyectos utilizaremos **PHP** con lenguaje del lado del servidor, las posibilidades son varias, y es posible utilizar **.ASP**, **CGI** o, incluso, archivos de texto.

Para tener acceso a los **métodos**, las **propiedades** y los **controladores de eventos**, es necesario crear una instancia de dicho objeto; esto se hace llamando al constructor del siguiente modo:

```
var miInstancia:LoadVars = new LoadVars();
```

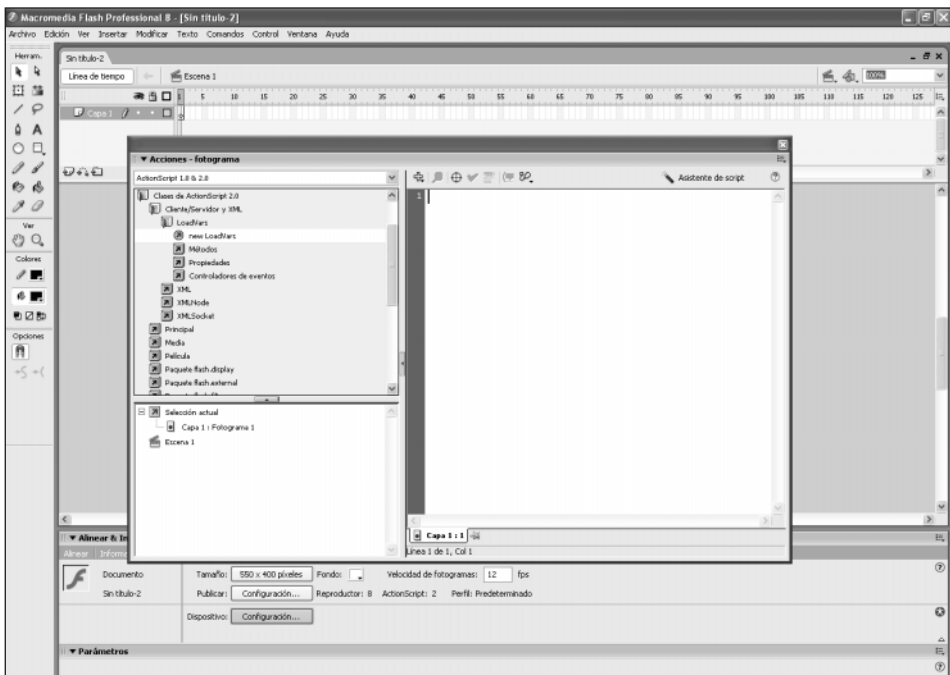


Figura 8. El objeto *LoadVars* se encuentra en la solapa *Clases* de *ActionScript 2.0*, dentro de la opción *Cliente/Servidor*.

III USO DE LOADVARIABLES

Antes de que **LoadVars** hiciera su aparición, se utilizaba el método **LoadVariables**, y muchos desarrolladores prefieren seguir empleándolo. Si bien en nuestros proyectos utilizaremos **LoadVars**, queda a criterio y comodidad de cada uno elegir qué método emplear; los resultados obtenidos serán los mismos. A modo de consejo, consideramos mejor el método **LoadVars**.

Métodos del objeto **LoadVars**:

load

send

sendAndLoad

getBytesLoaded()

getBytesTotal()

LoadVars.load

Sintaxis:

```
var miInstancia:LoadVars = newLoadVars();
miInstancia.load(url);
```

Disponibilidad:

ActionScript 1.0; Flash Player 6

Parámetros:

Url: Definimos el archivo del cual cargaremos las variables.

Explicación:

Utilizamos este método cuando queremos cargar desde **Flash** variables almacenadas en archivos externos. En nuestros ejemplos, dichas variables serán generadas por medio de **PHP**, pero como se especificó anteriormente, se pueden cargar contenidos generados con otros lenguajes, incluso variables almacenadas en archivos de texto plano o tabulados.

Ejemplo:

Desarrollaremos un pequeño ejemplo de cada uno de los métodos explicados para entender mejor su funcionamiento.

Los códigos utilizados son verdaderamente sencillos:



OTROS MÉTODOS

Para el desarrollo de nuestros proyectos, no será necesario el uso de los métodos **addRequest-Header**, **decode** y **toString**. De todos modos, en caso de necesitarlos, encontrará información acerca de esos métodos en la ayuda de **Flash**.

load.php

```
<?php
    echo "&laVariable= nuestra primera carga hecha desde un archivo externo";
?>
```

Generamos por medio de **PHP** una variable, que denominamos **laVariable**, y definimos su contenido, en este caso, una cadena de caracteres.

load fla

```
var miInstancia:LoadVars = new LoadVars();
miInstancia.load("load.php");
miInstancia.onLoad = armar;
function armar():Void{
    //this = miInstancia
    contenido = "contenido: " + this.laVariable;
}
```

En primer lugar, creamos una instancia del objeto **LoadVars**, que denominamos **miInstancia**, y luego aplicamos el método **load**, en el cual especificamos el parámetro **URL** asignando como valor la dirección **load.php**.

Utilizamos el controlador de eventos **onLoad**, el cual se invocará una vez cargado el contenido, y llamaremos a la función **armar**. Dentro de esa función, asignamos a la variable contenido (que se encuentra en nuestro escenario), el valor de la variable **laVariable**, generada por medio de **PHP**, como veremos en la siguiente imagen.

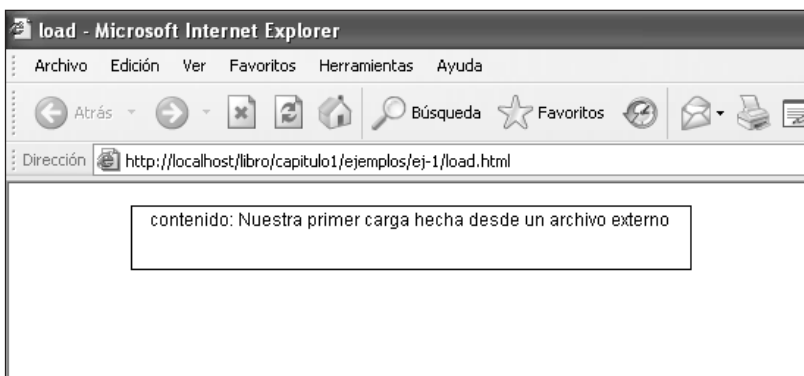


Figura 9. Nuestro primer ejemplo terminado.

LoadVars.send

Sintaxis:

```
var miInstancia:LoadVars = newLoadVars();
miInstancia.send(url[destino, método]);
```

Disponibilidad:

ActionScript 1.0; Flash Player 6

Parámetros:

Url: Definimos el archivo al cual enviaremos las variables.

Destino: Podemos definir la ventana en la cual recibiremos la respuesta del servidor. Su uso no es común en este método.

Método: Puede optar entre **Post** y **Get**. Por defecto se asigna el método **Post**.

Explicación:

Como su nombre lo indica, utilizamos el método **send** cuando deseamos enviar información de Flash hacia algún archivo externo.

Ejemplo:

En este sencillo ejemplo, enviaremos desde un formulario realizado en Flash cuatro variables a PHP, el cual las imprimirá en pantalla.

send fla

```
enviar.onRelease = function(){

    var miInstancia:LoadVars = new LoadVars();

    miInstancia.nombre = varNombre;
    miInstancia.apellido = varApellido;
    miInstancia.email = varEmail;
    miInstancia.comentarios = varComentarios;
    miInstancia.send("send.php","_blank");

}
```

En el escenario, tenemos cuatro variables y un botón para enviarlas. Sus nombres son **varNombre**, **varApellido**, **varEmail** y **varComentarios**.

En nuestra instancia del objeto **LoadVars**, tenemos cuatro variables, que llamamos **nombre**, **apellido**, **email** y **comentarios**.

Ahora bien, no habíamos desarrollado este tema hasta el momento, por lo cual le dedicaremos algunas líneas:

Al generar una instancia, podemos crear variables dentro de ella y asignarle valores. Como podemos ver, eso mismo es lo que hicimos en este sencillo ejemplo. Es importante que diferenciamos las variables que tenemos en el escenario de las que generamos dentro de **miInstancia**. Las que tenemos en el escenario son:

varNombre
varApellido
varEmail
varComentario

Las que generamos dentro de **miInstancia** son:

nombre
apellido
email
comentarios

Utilizamos las primeras para introducir los valores dentro de nuestra película de Flash; las otras son variables que creamos dentro de nuestra instancia y son las que enviaremos al archivo PHP utilizando el método **send**.

En definitiva, el contenido de la variable **nombre** será **varNombre**, y así sucesivamente con el resto de las variables.

Una vez aclarada esta diferencia, continuamos con nuestro código:

```
miInstancia.send("send.php", "_blank");
```



VARIABLES EXTERNAS PARA FLASH

Como verá, al generar la variable **laVariable** por medio de **PHP**, le asignamos el signo **&** al principio. Es importante que recuerde incluirlo siempre que vaya a cargar variables con **Flash** para que pueda reconocerlas como tales.

Aquí utilizamos el método **send** e indicamos dos parámetros: en primer lugar, el archivo al cual estamos enviando las variables (**send.php**), y luego definimos la ventana en la cual abriremos la información (**_blank**, abrirá una ventana nueva).



Figura 10. Nuestro formulario hecho en Flash.

send.php

```
<?php

echo "la información que recibimos desde Flash es la siguiente:<br>";

echo "nombre= " . $nombre . "<br>";
echo "apellido= " . $apellido . "<br>";
echo "e mail = " . $email . "<br>";
echo "comentarios = " . $comentarios . "<br>";

?>
```

Como podemos ver en el archivo **send.php**, imprimimos las variables que enviamos desde **Flash**.

AYUDA OFICIAL DE FLASH

Sitio web de **Adobe** donde encontrará información y ayuda respecto a **ActionScript**. La ayuda incluida en **Flash** también le puede ser de utilidad ante dudas que se le presenten. Accedemos al mismo desde www.macromedia.com/support/documentation/es/flashplayer/help/.

Por el momento, no complicaremos los ejemplos; la importancia aquí radica en que podamos diferenciar los distintos métodos del objeto **LoadVars** y sus posibilidades.

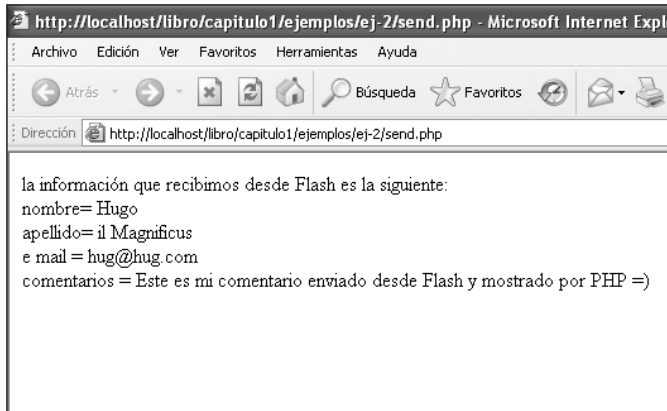


Figura 11. Nuestro archivo **PHP** con la información que enviamos desde **Flash**. Recuerde ejecutar los archivos en un servidor local o en Internet.

LoadVars.sendAndLoad

Sintaxis:

```
var miInstancia:LoadVars = new LoadVars();

miInstancia.sendAndLoad((url, objeto de destino, método);
```

Disponibilidad:

ActionScript 1.0; Flash Player 6

Parámetros:

Url: Definimos el archivo al cual enviaremos las variables.

Objeto de destino:

Es un objeto de **LoadVars** donde almacenaremos las variables que recibimos.

Método:

Puede optar entre **Post** y **Get**. Por defecto se asigna el método **Post**.

Explicación:

Por el nombre ya dedujimos cuál es el funcionamiento. Este método conjuga los dos métodos explicados anteriormente, y lo utilizamos para enviar y recibir datos.

Ejemplo:

Por el momento, no nos vamos a complicar con estos sencillos ejemplos; simplemente vamos a enviar una variable de **Flash** a **PHP** y luego la enviaremos de **PHP** a **Flash** con alguna información adicional.

La utilidad de este método es de un inmenso valor; podemos lograr una gran cantidad de aplicaciones, como validaciones de correos, consultas y muchísimas cosas más que iremos descubriendo a lo largo del libro.

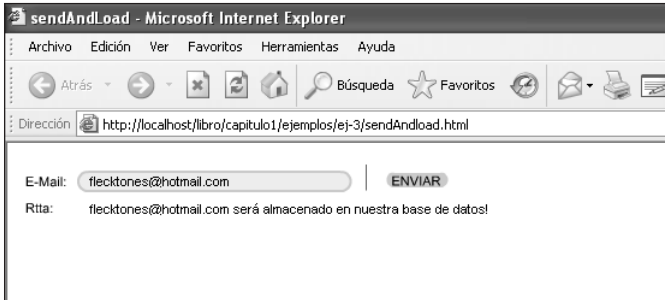


Figura 12. El archivo embebido dentro del Explorador. A través de éste, enviamos y recibimos variables.

sendAndLoad.fla

```

System.useCodepage = true;
boton.onRelease = function(){

    var enviamosInfo:LoadVars = new LoadVars();
    var recibimosInfo:LoadVars = new LoadVars();

    enviamosInfo.email = varEmail;
    enviamosInfo.sendAndLoad("sendAndload.php",recibimosInfo);

    recibimosInfo.onLoad = function(){
        validación = recibimosInfo.contenido;
    }
}

```

La diferencia con los ejemplos anteriores radica en que aquí creamos dos instancias del objeto **LoadVars**; utilizamos una de ellas para enviar la variable al archivo **sendAndLoad.php** y la otra para recibir en **Flash** la respuesta:

```
var enviamosInfo:LoadVars = new LoadVars();
var recibimosInfo:LoadVars = new LoadVars();
```

Nuestra otra línea nueva es la siguiente:

```
enviamosInfo.sendAndLoad("sendAndload.php",recibimosInfo);
```

Utilizamos una instancia para enviar las variables (en nuestro caso, la llamamos **enviamosInfo**) y definimos dentro de los parámetros del **sendAndLoad** el archivo al cual enviaremos las variables, y la instancia en la cual recibiremos la respuesta; a dicha instancia la llamamos **recibimosInfo**.

sendAndLoad.php

```
<?php
echo "&contenido=" . $email. " será almacenado en nuestra base de datos! ";
?>
```

Simplemente imprimimos una línea por medio del **echo** con la variable contenido; dicha variable es la que recibiremos en **Flash**.

Nuestro ejemplo hasta el momento no tiene utilidad, sólo estamos enviando y recibiendo variables de **Flash** a **PHP** y de **PHP** a **Flash**, pero podemos imaginar la innumerable cantidad de beneficios que nos brinda el objeto **LoadVars**.

LoadVars.getBytesTotal() y LoadVars.getBytesLoaded()

El método **getBytesTotal()**, como su nombre lo indica, nos devuelve la cantidad total en **Bytes** de los datos que cargaremos; y el método **getBytesLoaded()**, los **Bytes** cargados en el momento de ejecución de nuestra película. Estos métodos son de gran utilidad y no solamente en el uso del objeto **LoadVars**: también los emplearemos para cargas de imágenes y de sonidos, entre otras cosas. Conociendo estos valores, por medio de la regla de tres simple, podemos saber el porcentaje de los datos que han sido cargados.

A lo largo del libro, veremos sus usos.

Las propiedades con las que cuenta el objeto **LoadVars** son dos:

LoadVars.loaded

LoadVars.contentType

La primera nos devuelve un valor **booleano** que nos indica si una operación realizada por medio de los métodos **load** o **sendAndLoad** finalizó correctamente.

Por su parte, la propiedad **contentType** especifica el tipo **MIME** que se envía al servidor cuando llamamos a los métodos **send** o **sendAndLoad**. El valor predeterminado es **application/x-www-form-urlencoded**.

Controladores de eventos del objeto **LoadVars**:

onData

onHTTPStatus

onLoad

El evento **onData** se invoca cuando se han descargado completamente los datos del servidor o cuando se produce un error mientras se están descargando.

onHTTPStatus se invoca cuando la película recibe un código de estado **HTTP** del servidor; y el controlador **onLoad** se invoca cuando finaliza una operación del método **load()** o **sendAndLoad()**. Utilizaremos este método en reiteradas ocasiones, incluso, lo pusimos en funcionamiento en los ejemplos planteados anteriormente.

Objeto XML

La inclusión del objeto **XML** significó una inmensa gama de nuevas posibilidades para los desarrolladores, que pudieron lograr aplicaciones de una forma mucho más rápida e intuitiva. Sin duda es un modo sumamente prolijo de trabajar y de organizar la información. Dado que ya hemos explicado el objeto **LoadVars**, **XML** resultará verdaderamente sencillo; su funcionamiento también se basa en **propiedades**, **controlador de eventos** y **métodos**.

En versiones anteriores de referencia del lenguaje **ActionScript**, los **métodos** y las **propiedades** se incluían en la clase **XML**. Ahora se describen en la clase **XMLNode**.

XML

Aquí se listan las distintas características del objeto **XML**:

PROPIEDAD	DESCRIPCIÓN
ignoreWhite:Boolean	El valor predeterminado es false.
loaded:Boolean	La propiedad que indica si el documento XML se ha cargado correctamente.
status:Number	Establece automáticamente y devuelve un valor numérico que indica si un documento XML se ha analizado correctamente en un objeto XML.

Tabla 5. Propiedades del objeto **XML**.

EVENTO	DESCRIPCIÓN
<code>onData = function(src:String) {}</code>	Invocado cuando se ha descargado completamente el texto XML del servidor o cuando se produce un error mientras se está descargando.
<code>onLoad = function(success:Boolean) {}</code>	Lo invoca Flash Player cuando se recibe un documento XML del servidor.

Tabla 6. Eventos del objeto XML.

MÉTODO	DESCRIPCIÓN
<code>getBytesLoaded() : Number</code>	Devuelve el número de bytes cargados (sin interrupción) para el documento XML.
<code>getBytesTotal() : Number</code>	Devuelve el tamaño en bytes del documento XML.
<code>load(url:String) : Boolean</code>	Carga un documento XML de la URL especificada y sustituye el contenido del objeto XML especificado por los datos XML descargados.

Tabla 7. Métodos del objeto XML.

XMLNode:

En el siguiente cuadro, se muestran las propiedades de **XMLNode** y sus descripciones.

PROPIEDAD	DESCRIPCIÓN
<code>attributes:Object</code>	Objeto que contiene todos los atributos de la instancia XML especificada.
<code>childNodes:Array [read-only]</code>	Matriz de los elementos secundarios del objeto XML especificado.
<code>firstChild:XMLNode [read-only]</code>	Evalúa el objeto XML especificado y hace referencia al primer elemento secundario de la lista de elementos secundarios del nodo principal.
<code>lastChild:XMLNode [read-only]</code>	Valor XMLNode que hace referencia al último elemento secundario de la lista de elementos secundarios del nodo.
<code>nextSibling:XMLNode [read-only]</code>	Valor XMLNode que hace referencia al siguiente elemento secundario de la lista de elementos secundarios del nodo principal.
<code>nodeName:String</code>	Cadena que representa el nombre del nodo del objeto XML.
<code>nodeType:Number [read-only]</code>	Valor de <code>nodeType</code> , que puede ser 1 para un elemento XML o 3 para un nodo de texto.
<code>nodeValue:String</code>	Valor de nodo del objeto XML.
<code>parentNode:XMLNode [read-only]</code>	Valor de XMLNode que hace referencia al nodo principal del objeto XML especificado o devuelve Null si el nodo no tiene elemento principal.
<code>prefix:String [read-only]</code>	La parte del prefijo del nombre del nodo XML.
<code>previousSibling:XMLNode [read-only]</code>	Valor XMLNode que hace referencia al elemento secundario anterior de la lista de elementos secundarios del nodo principal.

Tabla 8. Propiedades de XMLNode.

Explicar cada uno de los métodos de forma separada y sin ver su funcionamiento práctico puede resultar poco útil; lo que haremos será utilizar un ejemplo en el cual explicaremos cada uno de los **métodos** y de las **propiedades** que vamos a utilizar a lo largo del libro, y otros que pueden resultar de importancia aunque no los utilicemos:

estructura.xml

```

<?xml version="1.0" encoding="iso-8859-1" ?>
- <contactos>
- <contacto>
  <personal nombre="marian" apellido="make" teléfono="4244305" edad="21"
    fecha="29/04/85" celular="15..." dirección="20 N.º 69" />
  <comentarios>éstos son los comentarios de makeeee.....!! !</comentarios>
</contacto>
- <contacto>
  <personal nombre="pepe" apellido="echeverry" teléfono="0303456" edad="40"
    fecha="10/04/66" celular="15..." dirección="diag. 80 N.º 2002" />
  <comentarios>éstos son los comentarios de pepe...</comentarios>
</contacto>
- <contacto>
  <personal nombre="eldita" apellido="busatti" teléfono="4223303" edad="55"
    fecha="10/09/51" celular="15..." dirección="mamá!! en casa!" />
  <comentarios>éstos son los comentarios de mamá...</comentarios>
</contacto>
- <contacto>
  <personal nombre="rodolfito" apellido="garofalo" teléfono="0221429382"
    edad="923" fecha="6/6/66" celular="15..." dirección="diag. 74 al fondo" />
  <comentarios>éstos son los comentarios de rodolfito...!</comentarios>
</contacto>
</contactos>

```

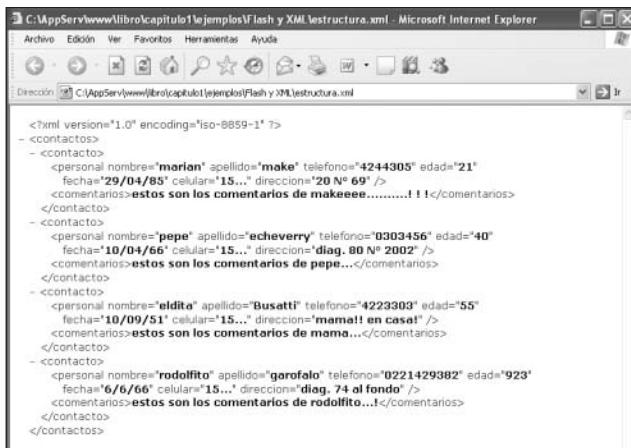


Figura 13. Visualización de la estructura XML en Internet Explorer.

xml fla

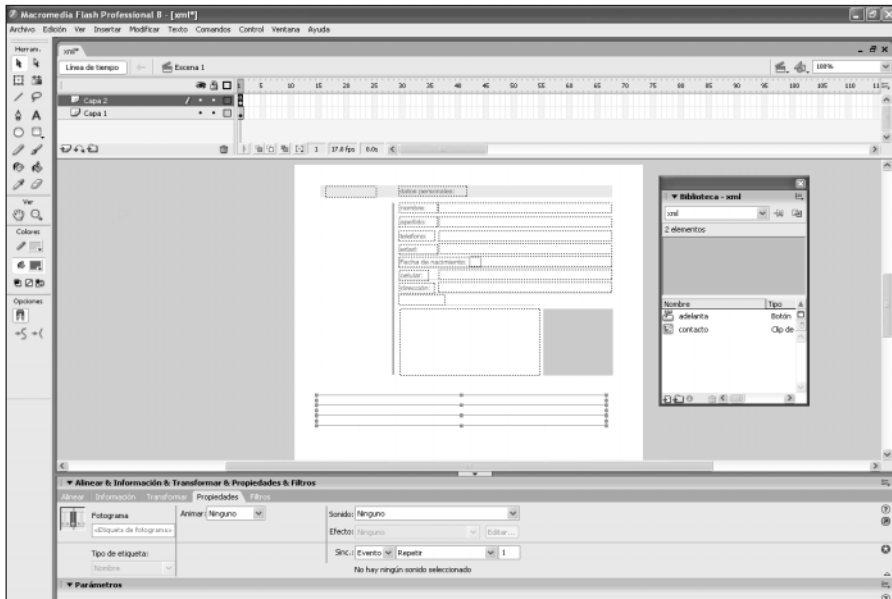


Figura 14. El escenario y la biblioteca del archivo *xml fla*.

```
System.useCodepage = true;
```

Habilitamos el uso de los caracteres del sistema operativo en el cual corre la película, para poder interpretar los archivos de texto externos.

```
var miXml:XML = new XML();
```

Al igual que con el objeto **LoadVars()**, para poder utilizar los métodos y las propiedades del objeto **XML**, debemos crear una instancia de dicho objeto. En este caso, llamaremos a nuestra instancia **miXml**.

SYSTEM.USECODEPAGE = TRUE;

Asignarle el valor **true** hace que utilicemos la página de códigos tradicionales del sistema operativo en el cual se ejecuta el reproductor para interpretar los archivos de texto externos. Debemos recordar incluir esta línea siempre, a fin de evitar inconvenientes con la letra Ñ, con los acentos y con los caracteres especiales.

```
miXml.ignoreWhite = true;
```

Al establecer el valor **true** en la propiedad **ignoreWhite**, lo que logramos es que los nodos de texto que contienen solamente espacios en blanco se descarten durante el proceso de análisis de la estructura.

Veremos que si comentamos dicha línea en el código del archivo **xml fla**, nuestro ejemplo dejará de funcionar.

```
miXml.load("estructura.xml");
```

Con el método **load**, lo que hacemos es indicar la ruta del archivo que vamos a abrir; en nuestro caso, cargaremos **estructura.xml**.

```
miXml.onLoad = function(success:Boolean) {
    if (success) {
        if (miXml.status == 0) {
            generarListado();
            estado = "ningún error. La estructura se cargó correctamente!";
        }
    }
}
```

Si la carga se efectuó correctamente, utilizamos la propiedad **status**, la cual devuelve un valor numérico que indica si el documento **XML** que se desea cargar se incluyó correctamente en el objeto **XML**. La siguiente lista presenta los valores que nos devuelve la propiedad **status**:

VALOR	DESCRIPCIÓN
0	No se ha producido ningún error; el análisis ha finalizado correctamente.
2	Una sección CDATA no se ha terminado correctamente.
3	La declaración XML no se ha terminado correctamente.
4	La declaración Doctype no se ha terminado correctamente.
5	Un comentario no se ha terminado correctamente.
6	Un elemento XML no tiene la forma correcta.
7	Memoria insuficiente.
8	El valor de un atributo no se ha terminado correctamente.
9	Una etiqueta start carece de la correspondiente etiqueta end.
10	Se ha encontrado una etiqueta end sin que exista una etiqueta start.

Tabla 9. Los valores numéricos devueltos por la propiedad **status** y la respectiva descripción de cada valor.

```
if (success) {
    if (miXml.status == 0) {
        generarListado();
        estado = "Ningún error. La estructura se cargó correctamente!";
    } else {
        switch (miXml.status) {
            case -2 :
                errorMessage = "Una sección CDATA no se ha terminado
                correctamente.";
                break;
            case -3 :
                errorMessage = "La declaración XML no se ha
                terminado correctamente.";
                break;
            case -4 :
                errorMessage = "La declaración DOCTYPE no se ha
                terminado correctamente.";
                break;
            case -5 :
                errorMessage = "Un comentario no se ha terminado
                correctamente.";
                break;
            case -6 :
                errorMessage = "Un elemento XML no tiene la forma
                correcta.";

                break;
            case -7 :
                errorMessage = "Memoria insuficiente.";
                break;
            case -8 :
                errorMessage = "El valor de un atributo no se ha
                terminado correctamente.";
                break;
            case -9 :
                errorMessage = "Una etiqueta start carece de la
                correspondiente etiqueta end. ";
                break;
            case -10 :
                errorMessage = "An end-tag was encountered without a
```

```

        matching start-tag.";
        break;
    default :
        errorMessage = "Se ha encontrado una etiqueta end
            sin que exista una etiqueta start.";
        break;
    }
}
};

```

Si el valor devuelto por **status** es 0, esto implica que la carga se efectuó correctamente, por lo cual llamamos **generarListado()** a la función que veremos a continuación:

```

function generarListado():Void {
    var cantidadDeNodos:Number = miXml.firstChild.childNodes.length;
}

```

Una vez que llamamos a la función, lo primero que hacemos es crear la variable del tipo numérico, que llamaremos **cantidadDeNodos**. Necesitamos averiguar la cantidad de nodos que contiene la estructura, ya que sobre la base de dicho valor, crearemos nuestra agenda personal.

Para averiguar dicho valor, en primer lugar, llamamos a nuestra instancia **miXml**. Luego accedemos al primer nodo de la instancia, que es el nodo padre de nuestra estructura **miXml.firstChild**.

La propiedad **firstChild** nos devuelve la primera raíz de nuestro objeto (nodo padre). Una vez que la conocemos, procedemos a preguntar por los nodos hijos de dicha estructura por medio de la propiedad **childNodes**.

miXml.firstChild.childNodes.

La propiedad **childNodes** nos devuelve los elementos secundarios del objeto **XML** especificado. Estos elementos secundarios son los que llamaremos nodos hijos.

miXml.firstChild.childNodes.length

Por último, la propiedad **length** nos devuelve el número de argumentos que se le pasan a la función; dicho de otro modo, evalúa la estructura y nos indica la cantidad de nodos hijos que ésta contiene.

En nuestro primer ejemplo, el valor de esta variable será 4, ya que es la cantidad de nodos hijos (**contacto**) con los que cuenta nuestro nodo principal (**contactos**):

```
var cantidadDeNodos:Number = miXml.firstChild.childNodes.length;
```

Una vez que conocemos el valor de la variable **cantidadDeNodos**, utilizamos dicho valor en el siguiente bucle **for**:

```
for (var i:Number = 0; i<cantidadDeNodos; i++) {
    var contacto:MovieClip = _root.attachMovie("contacto", "contacto"+i,
        _root.getNextHighestDepth());
    contacto._x = 38;
    contacto._y = 15*i+60;
```

Lo que hacemos aquí es un **attachMovie** del clip **contacto** que se encuentra en nuestra **biblioteca**, por cada vuelta que realice nuestro bucle (cuatro en total). Por ahora no nos adentraremos en estas cuestiones; llegado el momento, las analizaremos debidamente. Continuemos con **XML**.

A modo de resumen, hasta el momento, creamos una **instancia** del objeto **XML**, ignoramos los espacios en blanco de la estructura que íbamos a cargar, luego cargamos la estructura por medio del método **load** y comprobamos si la carga de la estructura se había realizado de forma correcta con la propiedad **Status**. Si el valor que nos devuelve **Status** es 0, todo indica que la estructura se cargó correctamente, y llamamos a la función **generarListado()**. Dentro de la función **generarListado()**, comenzamos a recorrer la estructura a fin de averiguar la cantidad de nodos hijos con los que cuenta. Conociendo este valor, generamos un bucle y hacemos un **attachMovie** de cada clip llamado **contacto** de nuestra **biblioteca** para cada nodo que exista en nuestra estructura.

Continuando, lo que debemos conseguir ahora, es crear variables dentro de los clips en cuestión, y a cada uno de ellos otorgarle un valor extrayéndolo de la estructura **XML** que cargamos. Vamos a ver de qué modo accedemos a cada nodo y cómo le asignamos cada valor:

```
contacto.nombre = miXml.firstChild.childNodes[i].firstChild.attributes.nombre;
```

Creamos la variable **nombre** dentro de cada uno de los clips que tenemos en el escenario. Para asignar el nombre, debemos extraerlo del siguiente modo:

```
contacto.nombre = miXml
```

Lógicamente, **miXml** irá siempre que llamemos a nuestra estructura; es decir, nuestra instancia, la cual habrá heredado, mediante las características de los objetos todos los **métodos**, **propiedades** y **controladores de eventos**.

```
contacto.nombre = miXml.firstChild
```

Del mismo modo en que lo hicimos hoy, con **firstChild** ingresamos al primer nodo de nuestra estructura; en nuestro caso, el nodo padre.

```
contacto.nombre = miXml.firstChild.childNodes[i]
```

Pocas líneas atrás, explicamos la propiedad **childNodes** y dijimos que nos devolvía los elementos secundarios del objeto **XML** especificado; es decir, cada uno de nuestros nodos hijos como objetos del mismo tipo.

Ahora bien, para ingresar a cada uno de dichos nodos, debemos hacerlo utilizando la variable **i** que empleamos en el bucle **for**; esto implica que por cada vuelta que realice nuestro bucle, indicará a qué nodo hijo estamos haciendo referencia.

```
contacto.nombre = miXml.firstChild.childNodes[i].firstChild
```

Dentro de cada nodo hijo (**contacto**), tenemos dos nodos nietos:

personal

comentarios

Nosotros debemos averiguar el nombre del contacto, y dicha información se encuentra dentro del nodo nieto (hijo del hijo) **personal**; por eso, utilizamos **firstChild** para ingresar a dicho nodo.

```
contacto.nombre = miXml.firstChild.childNodes[i].firstChild.attributes.nombre
```

Aquí encontramos una nueva propiedad (**attributes**) que contiene todos los atributos de la instancia **XML** especificada. Podríamos considerar estos atributos como propiedades del objeto.

Si analizamos nuestra estructura, **nombre**, **apellido**, **teléfono**, **edad**, **fecha**, **celular** y **dirección** son atributos del nodo nieto **personal**.

Para acceder a ellos, utilizamos la propiedad **attributes** del objeto **XML** y luego indicamos el nombre de dicho atributo; en nuestro caso, deseamos averiguar el nombre.

Si analizamos nuestras próximas líneas de código, veremos que estamos haciendo lo mismo que hicimos hasta el momento, sólo que cambiamos el nombre del atributo del cual deseamos extraer la información:

```

contacto.apellido = miXml.firstChild.childNodes[i].firstChild.attributes.apellido;

contacto.teléfono = miXml.firstChild.childNodes[i].firstChild.attributes.teléfono;

contacto.edad = miXml.firstChild.childNodes[i].firstChild.attributes.edad;

contacto.fechaNac = miXml.firstChild.childNodes[i].firstChild.attributes.fecha;

contacto.celular = miXml.firstChild.childNodes[i].firstChild.attributes.celular;

contacto.dire = miXml.firstChild.childNodes[i].firstChild.attributes.dirección;

```

Si bien ya tenemos todos los datos que necesitamos de cada uno de nuestros contactos, vamos a crear algunas variables más para seguir viendo propiedades del objeto:

```

contacto.comentarios = miXml.firstChild.childNodes[i].firstChild.nextSibling.
firstChild.nodeValue;

```

Aquí tenemos dos nuevas propiedades; **nextSibling** y **nodeValue**. Vamos a verlas:

contacto.comentarios = miXml.firstChild.childNodes[i].firstChild

Hasta aquí, nada nuevo. Vimos hoy que utilizamos la misma ruta para acceder al nodo nieto **personal**. Ahora bien, dijimos que tenemos dos nietos: **personal** y **comentarios**. Para asignar los comentarios, debemos bajar un nivel en nuestra estructura, ya que el nodo nieto **comentarios** (al cual tenemos que acceder) se encuentra por debajo del nodo nieto **personal**.

Para esto utilizamos **nextSibling**, que hace referencia al siguiente elemento secundario de la lista de elementos secundarios del nodo principal

contacto.comentarios = miXml.firstChild.childNodes[i].firstChild.nextSibling.firstChild

Es importante prestarle atención al **firstChild** que se ubica luego de **nextSibling**; ya que si bien con **nextSibling** nos ubicamos en el nodo indicado, para ingresar a él debemos hacerlo con **firstChild**.

```

contacto.comentarios = miXml.firstChild.childNodes[i].firstChild.nextSibling.
firstChild.nodeValue;

```

Una vez que ingresamos al nodo, averiguamos su contenido por medio de la propie-

dad **nodeValue**. Si el nodo al cual nos referimos es un nodo de tipo texto, **nodeValue** es el contenido de dicho nodo.

Es importante diferenciar un **atributo** del **valor** de un nodo:

nodeValue:

```
<ejemplo> Esto que estás leyendo es el valor de un nodo, y para ingresar a él
tenés que hacerlo con nodeValue =) </ejemplo>
```

attributes y nodeValue:

```
<ejemplo atributo = "esto es un atributo!"> el valor del nodo es el que estás
leyendo ahora...</ejemplo>
```

Los atributos del nodo (**attributes**) se encuentran dentro de la etiqueta de apertura, mientras que los valores (**nodeValue**) se encuentran entre la etiqueta de apertura y la etiqueta de cierre.

Ahora bien, otra posibilidad es que nos interese saber el nombre de la etiqueta; podemos averiguarlo con la propiedad **nodeName**.

En el escenario del archivo **xml fla**, veremos que a la izquierda tenemos las variables que nos indican el nombre, el apellido, etc. Si observamos la última de dichas variables, veremos que la llamamos **varComentarios** y que, a diferencia de las demás, no escribimos su valor en dicho campo de texto, sino que extraemos su valor de la estructura **XML**:

```
varComentarios = miXml.firstChild.childNodes[i].firstChild.nextSibling.nodeName+ " :";
```

nodeName nos devuelve el nombre de una etiqueta.

```
<uno> el nodeName de este nodo es uno</uno>
<dos> el nodeName de este nodo es dos</dos>
```

Así como hace instantes explicamos la propiedad **nextSibling** para acceder al próximo nodo de nuestra estructura, también podemos utilizar **previousSibling** para acceder a la posición anterior:

```
contacto.posterior = miXml.firstChild.childNodes[i].nextSibling.firstChild.
```

```

attributes.nombre;
contacto.anterior = miXml.firstChild.childNodes[i].previousSibling.firstChild.

attributes.nombre;

```

Con estas dos líneas, averiguamos el nombre del contacto anterior y el nodo del contacto siguiente respecto al nodo que seleccionamos.

Por último, veremos la propiedad **lastChild**:

contacto.último = miXml.firstChild.lastChild.firstChild.attributes.nombre;
lastChild hace referencia al último elemento secundario de la lista de elementos secundarios del nodo.

Una vez que asignamos todos los valores a las variables de los movie clips, las aplicamos a los campos de texto dinámicos que se encuentran en el escenario al presionar sobre cada uno de ellos:

```

contacto.onRelease = function() {
    nombre = this.nombre;
    apellido = this.apellido;
    teléfono = this.teléfono;

    edad = this.edad;
    fecha = this.fechaNac;
    celular = this.celular;

    dirección = this.dire;
    comentarios = this.comentarios;
    if (this.anterior == undefined) {
        this.anterior = "nadie";
    } else if (this.posterior == undefined) {
        this.posterior = "nadie más";
    }
    ubicación = "el contacto "+this.nombre+" se encuentra entre
        "+this.anterior+" y "+this.posterior;
    primero = "el primer contacto es " +this.primer0
    último = "el último contacto es " + this.último;
};

```

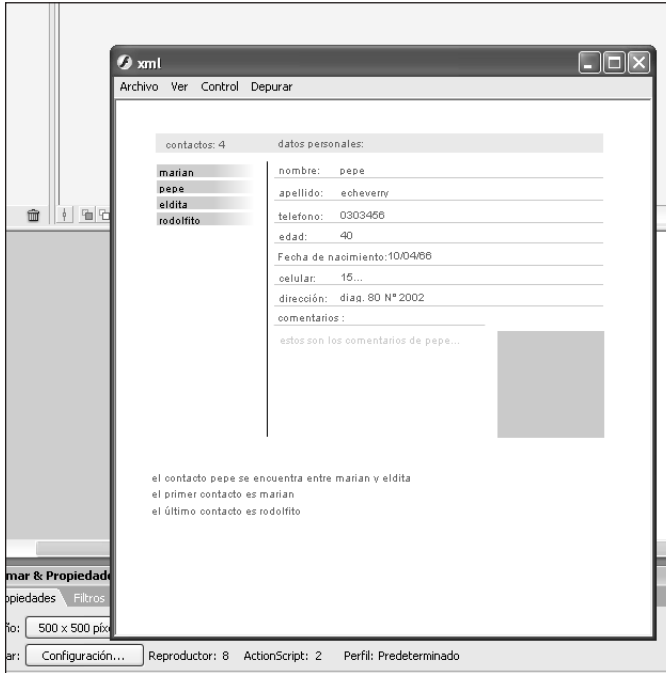


Figura 15. Archivo xml fla.

CONCLUSIÓN

Haremos un repaso de los conceptos adquiridos durante el desarrollo del capítulo.

Base de datos: Ya sabemos cuál es su finalidad, y nos es imprescindible para los proyectos que desarrollaremos de aquí en adelante. Al trabajar de forma dinámica con datos, necesitamos de un soporte digital que nos sirva para almacenar esa información que mostraremos, agregaremos, borraremos, etc. Para esto mismo, usaremos las bases de datos.

PHP: PHP es el lenguaje del lado del servidor que cumplirá dos funciones principales; la de almacenar datos en la base de datos, y la de extraerlos de esa base y generar con dicha información estructuras XML.

XML: Es de vital importancia para nuestros proyectos; es el nexo entre el cliente y el servidor; es lo que genera PHP sobre la base de la información almacenada en la base de datos para que interprete Flash.

Flash: Es el lado del cliente de nuestros proyectos, donde sobre la base de las estructuras XML que se van a cargar, generamos la interfaz gráfica de cada desarrollo.

También enviaremos variables a **PHP** desde **Flash**, por medio del método **SendAndLoad** del objeto **LoadVars**.

Flash, ¿LoadVars o XML?

Probablemente nos preguntaremos cuál es la verdadera diferencia entre **XML** y el objeto **LoadVars**, y es interesante aclarar esta duda. Utilizaremos **LoadVars** para el envío y la recepción de variables sencillas, sin estructuración. Su uso es recomendable para, por ejemplo, mostrar una frase que indica la cantidad de visitantes o de usuarios conectados a nuestro sitio. En cambio, utilizaremos **XML** para la carga de datos más bien complejos, que requieren una estructura. La diferencia principal radica en la complejidad de los datos que manejan uno y otro; **LoadVars** es más que útil para el envío o la recepción de variables más bien sencillas, y **XML** es imprescindible para la carga de datos estructurados, con jerarquías. En los ejemplos, veremos que utilizamos **XML** para carga de estructuras, y **LoadVars** para el envío de variables.

RESUMEN

Visto este primer capítulo introductorio en el cual mencionamos las particularidades necesarias de cada lenguaje para nuestros proyectos, pasaremos al capítulo dos, donde comenzaremos a utilizarlos simultáneamente e iremos viendo las distintas aplicaciones que surjan de la combinación de estas tecnologías.



ACTIVIDADES

TEÓRICAS

- 1 ¿Para qué utilizamos PHP?

- 2 ¿Qué es XML?

- 3 ¿Cómo está compuesta una estructura XML?

- 4 ¿De qué modo ingresamos un nuevo registro a la base de datos?

- 5 ¿De qué imprimimos un resultado de la base de datos?

- 6 ¿Con qué objeto de Flash cargamos estructuras XML?

- 7 ¿Cómo enviamos variables desde Flash a PHP?

- 8 ¿Cuál es la diferencia entre los objetos XML y LoadVars?

PRÁCTICAS

- ✓ Cree una nueva base de datos.

- ✓ Almacene información de cualquier tipo dentro de ella.

- ✓ Utilice PHP para imprimir una estructura XML con los contenidos de dicha base de datos.

- ✓ Utilice Flash para cargar dicha estructura.

- ✓ Sirviéndose del pequeño ejemplo de la lista de contactos realizada con Flash y XML, genere alguna aplicación similar a ésta, utilizando las propiedades y métodos del objeto XML.
