USERS

INCLUYE PROYECTOS REALES ELABORADOS PASO A PASO

FLASH EXTREMO

TODO EL PODER DE FLASH CS4 Y ACTIONSCRIPT 3.0

INTEGRACIÓN DE FLASH CON APIS Y PHP

CREACIÓN DE APLICACIONES CON ADOBE AIR

FLASH MEDIA SERVER 3 Y GRABACIÓN DE VIDEO

CONTENIDOS EXTERNOS: IMÁGENES, VIDEOS Y SONIDOS

REPRESENTACIONES VISUALES DE SONIDOS

por MARIANO MAKEDONSKY

DESARROLLO DE APLICACIONES DE ALTO IMPACTO

TÍTULO: FLASH EXTREMO

AUTOR: Mariano Makedonsky

COLECCIÓN: Manuales USERS

FORMATO: 17 x 24 cm

PÁGINAS: 320

ISBN 978-987-663-009-2

Makedonsky, Mariano
Flash extremo. - 1a ed. - Banfield - Lomas de Zamora : Gradi, 2009.
320 p. ; 24x17 cm. - (Manual users; 174)

ISBN 978-987-663-009-2

1. Informática. I. Título
CDD 005.3

Léalo antes Gratis!

En nuestro sitio puede obtener, en forma gratuita, un capítulo de cada uno de los libros: **redusers.com**

RedUSERS
COMUNIDAD DE TECNOLOGÍA

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios, glosarios, atajos de teclado y todos los elementos necesarios para asegurar un aprendizaje exitoso y estar conectado con el mundo de la tecnología.

Conéctese con nosotros

> **ARGENTINA** ✆ (011) 4110.8700 | **CHILE** ✆ (2) 810.7400 | **ESPAÑA** ✆ (93) 635.4120

✉ **usershop@redusers.com**

redusers.com

Mariano Makedonsky

Abocado al diseño y a la multimedia, se dedica al desarrollo de aplicaciones dinámicas integrando Flash, ActionScript 3.0, Adobe AIR, PHP, MySQL, XML, Flash Media Server 3, RED5, entre otras tecnologías, en sus proyectos.
Este es el segundo libro del autor para la editorial. Anteriormente escribió *PROYECTOS WEB: Flash + PHP + MySQL + XML.*

Agradecimientos

A Fabricio Mouzo, por haber escrito el capítulo 9 de este libro y compartir en él toda su experiencia en el desarrollo de aplicaciones de escritorio con Adobe AIR. También por el constante aporte de sus conocimientos y ayuda en esta obra y por compartir este fanatismo desmedido por Flash y por la multimedia.

A Julio, Gra, el Negro y Rosa.
A Carlos, Lili, Seba y Pau.
A los de siempre: Nico, Bety, Cabe, Artu, Fer, Fran, Maio.
A Ivi, por su paciencia y cariño.
A la gente de multimedia: Topo, Fa, NaN, Fer V, Ale C, Ale I, Andy, Bru, Sanfe, Facu C, Fer Amenta, Fer Rodriguez Arias, Pato M, Pato O.
A Diego y Pau, por su profesionalidad para guiarme durante el desarrollo de esta obra.
A vos, por dedicar un minuto de tu tiempo a esta parte del libro. ¡Espero que disfrutes de esta obra!

Dedicatoria

A mi vieja y a mi hermano; sin ellos, esta obra no sería posible.

PRÓLOGO

Cuando comencé a dar mis primeros pasos con Flash (hace ya algunos años, allá por la versión 5 y luego MX del programa), pocos creían con firmeza que la plataforma pudiese llegar a evolucionar y dejar de lado su condición de "programa para animaciones pesadas, cansadoras e incargables". Flash ha tenido durante mucho tiempo una gran cantidad de detractores, y si bien algunos de los cuestionamientos eran lógicos y aceptables, otros carecían completamente de fundamentos. En definitiva, con razón o no, se fue moldeando un escepticismo respecto a Flash.

Pero también hubo quienes creyeron y consideraron que Flash era mucho más que un programa de animación vectorial, y entendieron que debía experimentar una lógica evolución. Lentamente, se fue gestando una comunidad de usuarios que logró ver a Flash desde otra óptica. Y la propia plataforma, evolución mediante, le brindó a esa comunidad la tecnología ActionScript, que permitió mucho más que animaciones. Producto de este continuo avance, aquello que en algún momento fue una herramienta de animación vectorial hoy es una plataforma profesional para llevar a cabo casi cualquier tipo de desarrollo multimedial y que a su vez cuenta con un lenguaje de programación sólido, estable y serio que nos posibilita un amplio espectro de soluciones: ActionScript 3.0.

Tengo el honor de haber escrito estas páginas, porque son en gran parte el reflejo de mi camino de aprendizaje y mi experiencia, que se fue forjando gracias a la comunidad de desarrolladores y a todos aquellos que creyeron en Flash, compartieron constantemente sus conocimientos; y supieron entender que no se trata de buenas o malas plataformas, sino de buenos o malos usos que se hace de ellas. Hoy, debido a todos aquellos que confiaron, los sitios web 2.0 de mayor relevancia en el mundo delegan gran parte de sus contenidos multimediales a Flash, y el plugin de Flash Player es el de mayor penetración a nivel mundial.

Deseo que todo aquello que plasmé en este libro pueda ser capitalizado y devuelto a la comunidad en la forma que realmente merece: grandes desarrollos que evidencien el potencial de Flash.

Mariano Makedonsky
mariano.makedonsky@gmail.com

EL LIBRO DE UN VISTAZO

Esta obra presenta un recorrido por los temas más apasionantes que proponen la plataforma Flash CS4 y su lenguaje ActionScript 3.0. Veamos una breve descripción de los temas que abordamos en cada uno de los capítulos.

Capítulo 1

INTRODUCCIÓN A FLASH CS4

Flash CS4 propone una serie de ventajas e incorporaciones respecto a sus versiones antecesoras y las analizaremos a lo largo de este capítulo. Haremos un recorrido por las nuevas vistas del programa y conoceremos las últimas herramientas: Huesos y 3D.

Capítulo 2

INTRODUCCIÓN A ACTIONSCRIPT 3.0

Haremos un acercamiento ameno a la sintaxis de ActionScript 3.0, la última versión del lenguaje, derribando mitos y temores respecto a su complejidad. Presentaremos comparaciones con AS 1.0 y 2.0 para quienes están migrando a AS 3.0, y una breve introducción a la Programación Orientada a Objetos (OOP).

Capítulo 3

DIBUJO EN FLASH

En este capítulo proponemos una introducción al dibujo por código en AS 3.0 y el análisis de sus ventajas respecto al dibujo desde la IDE de Flash. Luego utilizaremos lo aprendido para generar una pizarra de dibujo e integrarla con PHP para almacenar los dibujos como imágenes JPG en un servidor.

Capítulo 4

DESARROLLO DE FORMULARIOS

Haremos un recorrido en profundidad para lograr un mejor manejo de los campos de textos y veremos varios conceptos de importancia que contribuyen a la usabilidad en desarrollos que impliquen el uso de texto en Flash. Finalmente, integraremos el desarrollo del formulario con PHP para poder enviar los contenidos vía e-mail.

Capítulo 5

GALERÍA Y BUSCADOR DE IMÁGENES EN FLASH

Realizaremos el abordaje de los conceptos necesarios para manejar imágenes con fluidez, a través del desarrollo de una galería dinámica por medio de la carga de estructuras XML. Finalmente veremos de qué manera integrar Flash con la API de flickr para crear un buscador de imágenes.

Capítulo 6

SONIDOS EN FLASH Y ESPECTROS DE SONIDO

Abarcaremos la nueva sintaxis que ActionScript 3.0 propone para el manejo de sonidos y concluiremos con el desarrollo de un reproductor de MP3, tomando los archivos de forma externa. También abordaremos el nuevo método computeSpectrum() para generar representaciones visuales de los sonidos.

Capítulo 7

VIDEO EN FLASH

Veremos los conceptos necesarios para generar archivos FLV y cargarlos de manera

interna o externa. Conoceremos la sintaxis necesaria para crear un reproductor de videos completamente personalizado y exploraremos las nuevas ventajas que AS3 propone para reproducir videos en modo de pantalla completa (fullscreen).

Capítulo 8

MICRÓFONO, CÁMARA WEB Y FLASH MEDIA SERVER 3

Abordaremos la sintaxis para manipular micrófonos y webcams; cómo tomar una imagen de una cámara web, cómo enviarla al servidor en formato JPG o realizar desarrollos experimentales. Concluiremos integrando a Flash con Flash Media Server 3 para grabar videos desde Flash y almacenarlos en formato FLV.

Capítulo 9

DESARROLLOS CON ADOBE AIR

Haremos una introducción a Adobe AIR y a su sintaxis. Gracias a su implementación, podemos generar aplicaciones de escritorio integrando AIR con Flash. Veremos sus ventajas a través de ejemplos prácticos, creando un navegador web y generando un reproductor de MP3 para el escritorio.

Apéndice

MANEJO DE ARCHIVOS

En este apéndice encontraremos lo que necesitamos para trabajar con la subida de archivos en nuestras aplicaciones Flash. Para eso, existen clases especiales que aprenderemos a utilizar, tanto para subir un archivo como varios al mismo tiempo.

Servicios al lector

En esta última sección brindamos un índice que nos ayudará a encontrar de forma rápida los términos más importantes de esta obra. Además, veremos un listado de sitios de interés para ampliar nuestros conocimientos y mantenernos al tanto de las últimas novedades en la materia.

! INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual encontrará una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados.
Cada recuadro está identificado con uno de los siguientes iconos:


CURIOSIDADES E IDEAS


ATENCIÓN


DATOS ÚTILES Y NOVEDADES


SITIOS WEB

UNA NUEVA DIMENSIÓN

RedUSERS
COMUNIDAD DE TECNOLOGÍA

MATERIAL ADICIONAL

Ejemplos, código fuente, planillas y otros elementos para descargar. Mejoran su experiencia de lectura y le ahorran tiempo de tipeado.

GUÍA

Una completa guía con sitios web, para acceder a más información y recursos útiles que le permitirán profundizar sus conocimientos.

SOFTWARE

Las mejores aplicaciones, relacionadas con el contenido del libro, comentadas y listas para bajar.

FOROS

Le permitirán realizar intercambios de dudas, respuestas y opciones con otros lectores y estar en contacto con especialistas de la editorial.

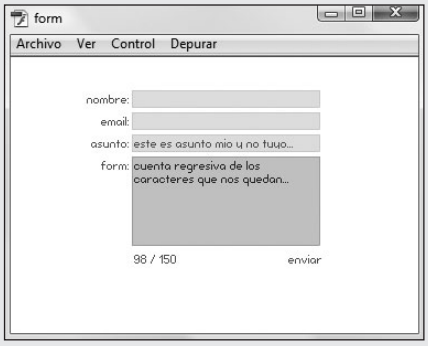
CAPÍTULO GRATIS

No compre a ciegas. De cada título, ponemos un capítulo para descarga gratuita. Evalúe nuestros libros antes de decidir su compra.

redusers.com

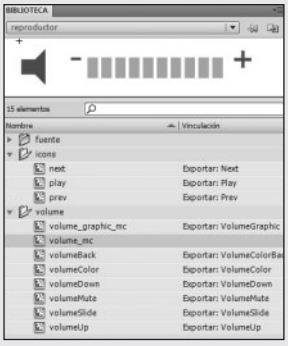
CONTENIDO

Capítulo 4



Capítulo 5

Capítulo 6



INTRODUCCIÓN

Flash y su lenguaje ActionScript han sufrido variaciones a lo largo de su historia que hacen que, al momento de escribir estas líneas, sea complejo limitar el entorno de desarrollo que ambos proponen. Hay un factor determinante para que esto sea así: la plataforma evolucionó a pasos agigantados y lo que en sus comienzos era un software destinado a la animación vectorial, hoy es un entorno de desarrollo profesional que brinda posibilidades para animadores, diseñadores de comunicación visual y multimediales, programadores y artistas, entre otras ramas. Esta convergencia de diversas disciplinas hace de Flash lo que es, y difícilmente se logre una mejor evolución para esta plataforma y lo que se obtiene como resultado de la intervención multidisciplinaria.

Pero a su vez, la principal virtud genera una serie de dudas respecto a Flash y ActionScript 3; haber sido expuesto a tan notoria evolución, deviene en cambios en el modo de desarrollar, diseñar y animar, lo que instaura miedos, confusiones y mitos.

Este libro surge para desmitificar conceptos erróneos respecto a Flash y a ActionScript 3 y para conocer los nuevos horizontes que esta tecnología nos propone en combinación con otras, como PHP, Flash Media Server 3, Adobe AIR y APIs.

El hecho de contar con un púbico tan heterogéneo, nos hizo replantear el enfoque que daríamos a este libro; que puede ser de utilidad tanto para quienes están comenzando a abordar Flash CS4 y AS3, como para aquellos desarrolladores avanzados en la materia. Para lograrlo, dividimos los capítulos prácticos en dos partes; una primera orientada a sentar conceptos básicos que hacen a Flash y a ActionScript 3 y una segunda donde abarcamos temas más avanzados de utilidad para los desarrolladores, que son un fiel reflejo de las aplicaciones que requiere actualmente la industria multimedial.

Y porque consideramos que generar sentido de comunidad es lo que nos hace enriquecernos con nuestras experiencias, cualquier duda, consulta o sugerencia será bienvenida en el grupo ActionScript Hispano en Facebook (**www.facebook.com**) y en el sitio **http://as3hispano.com**.

Introducción

Este primer capítulo tiene por finalidad presentar las mejoras del entorno gráfico de Flash en su versión CS4. Si bien los principales cambios están dados en el campo de la animación y el diseño, también veremos otros progresos y aspectos que facilitan nuestro desempeño en el desarrollo en Flash.

INTRODUCCIÓN A FLASH CS4

En su última versión, Flash presenta una serie de mejoras que, si bien están principalmente orientadas al entorno de diseñadores y animadores, es un aspecto en común que nos incumbe a todos los miembros de la comunidad Flash. Seamos diseñadores o desarrolladores, el entorno gráfico de este programa es nuestro punto de partida a la hora de comenzar nuestros proyectos. En esta breve introducción veremos las principales características y progresos de **Flash CS4**.

Entorno gráfico y distribución de contenidos

Para todo aquel desarrollador o diseñador que ha utilizado las versiones anteriores de Flash, probablemente haya sido una sorpresa el haber abierto Adobe Flash CS4. A primera vista, encontraremos un entorno totalmente renovado y con una nueva distribución de los contenidos.

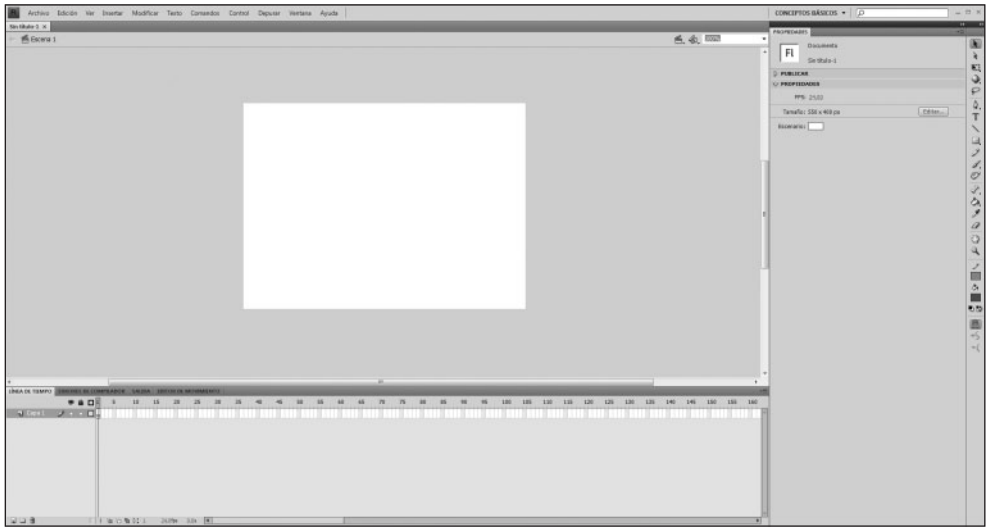


Figura 1. *Nuevo entorno gráfico de* ***Flash CS4****, con una distribución de paneles diferente.*

DESCARGA DE FLASH CS4

En caso de no disponer de la versión **CS4** de **Flash**, podemos bajar la versión trial del software desde el sitio oficial de Adobe (**www.adobe.com/downloads**). Para hacer la descarga, deberemos iniciar sesión con un Adobe ID, que es posible obtener con un simple registro.

Dentro de esta nueva distribución, seguramente sea la **línea de tiempo** (**timeline**) lo que más llamará nuestra atención. Ésta se encuentra abajo y no arriba como sucedía en todas las versiones anteriores de Flash. De todos modos, el entorno es totalmente personalizable. Simplemente, es cuestión de adaptar los paneles a nuestras preferencias. Otro cambio llamativo es la nueva ubicación del **panel de propiedades**, que anteriormente se hallaba en el lugar que actualmente ocupa la línea de tiempo. Esto es, en verdad, una gran ventaja en vista a las nuevas opciones que se le han agregado, ya que su manejo se tornaría un tanto confuso al tenerlo en la parte inferior. En su ubicación actual es más fácil la lectura de las opciones con las que cuenta este panel, las cuales no siempre son las mismas sino que varían de acuerdo a la acción que estemos realizando.

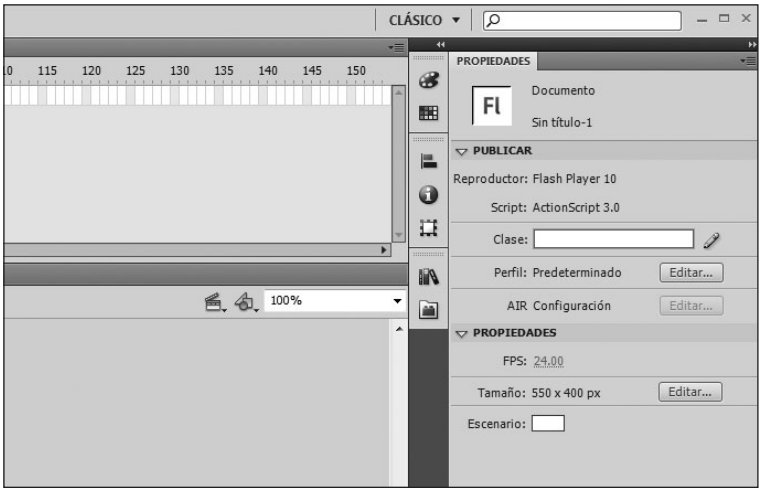


***Figura 2.** Nueva ubicación y distribución del **panel PROPIEDADES**. Esta posición hace más legible sus contenidos y nos brinda un mayor espacio para la escena **IDE** de Flash.*

Más allá de que el entorno gráfico sea totalmente personalizable, es interesante saber que en el borde superior derecho contamos con un menú llamado **CONCEPTOS BÁSICOS**. Al hacer clic en él, aparece una lista de **espacios de trabajo** preestablecidos. De esta

III DEFINICIÓN DE IDE

Es importante mencionar que emplearemos el término **IDE** en reiteradas oportunidades a lo largo del libro. IDE es el acrónimo de **Integrated Development Environment** (en español, **Entorno Integrado de Desarrollo**). Al nombrar la IDE de Flash estamos haciendo mención a la interfaz gráfica de usuario del programa (**GUI**, **Graphical User Interface**).

manera, Flash genera una distribución que se adapta mejor a las necesidades del tipo de tarea que realizamos, generando un determinado espacio de trabajo. Por ejemplo, al optar por la opción **Animador**, la línea de tiempo vuelve al lugar que tenía en las versiones anteriores y se distribuye una serie de paneles cuya utilidad está relacionada con la animación de contenidos. En cambio, con la opción **Desarrollador** se elimina la línea de tiempo, de la que generalmente prescindimos, y contamos con el panel **SALIDA**, el panel **PROYECTO** y el de **COMPONENTES** de Flash.

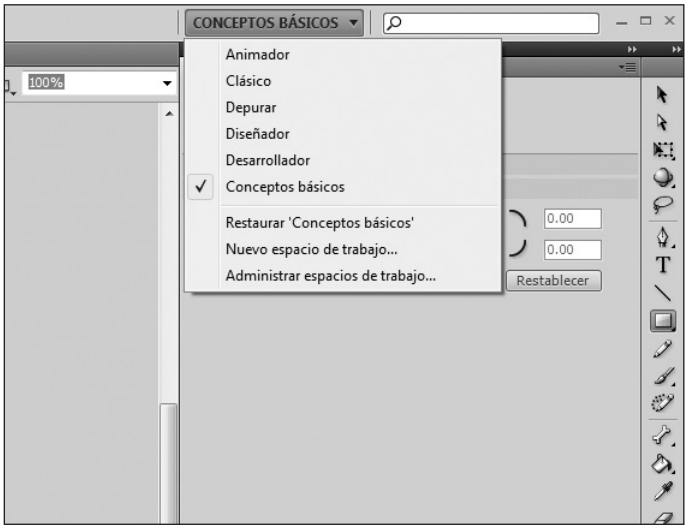


***Figura 3.** Dentro del menú **CONCEPTOS BÁSICOS**, encontramos una serie de configuraciones preestablecidas del entorno gráfico. Cada una de éstas se adapta mejor según la tarea que vamos a realizar.*

Si bien estos cambios son de utilidad, tienen mucho que ver las costumbres de quien está utilizando el programa. Seguramente, le resulte muy complicado acostumbrarse a esta nueva línea de tiempo a aquel diseñador o desarrollador que utiliza Flash desde sus primeras versiones. Por esta razón, la mejor alternativa a la hora de definir nuestro propio espacio de trabajo es configurar los paneles de manera tal que la distribución sea la adecuada a nuestras necesidades.

III EXTENSIÓN DE ADOBE KULER PARA FLASH CS4

Otra de las grandes novedades de Flash CS4 es contar con la extensión de **Kuler** (aplicación para el manejo de colores y paletas de colores). Para abrirlo, debemos ir a **Ventana/Extensiones/Kuler**. También podemos visitar su sitio web en **http://kuler.adobe.com/**.

Línea de tiempo (timeline)

La línea de tiempo es, sin lugar a dudas, el lugar donde vamos a encontrar la mayor cantidad de cambios. Para quienes disfruten de la animación en Flash en **timeline**, definitivamente la versión CS4 es la indicada: hay una serie de modificaciones en el modo de animar que hacen la tarea mucho más completa.

En primer lugar, en Flash CS4 **animamos objetos**. Este es un nuevo tipo de animación donde cada objeto sobre el escenario cuenta con su propia línea de tiempo, su capa y su sistema de animación. Para crear una animación en Flash CS4, primero debemos generar un objeto utilizando el panel **HERRAMIENTAS**. Una vez que posicionamos el objeto sobre el escenario, hacemos clic con el botón derecho sobre el **frame** que contiene el objeto y presionamos **Crear interpolación de movimiento**.

***Figura 4.** Primero creamos la **interpolación** y, luego, deberemos definir dónde finaliza.*

III ANIMACIÓN EN TIMELINE VERSUS ANIMACIÓN POR CÓDIGO

En Flash, tenemos dos maneras de animar: desde la línea de tiempo (**timeline**) o con ActionScript. Generalmente, se emplea **la animación en timeline** para animaciones estáticas y se utiliza **la animación por medio de código** para animaciones interactivas que requieran de comportamientos aleatorios (**random**) o que tengan una complejidad difícil de representar en la línea de tiempo.

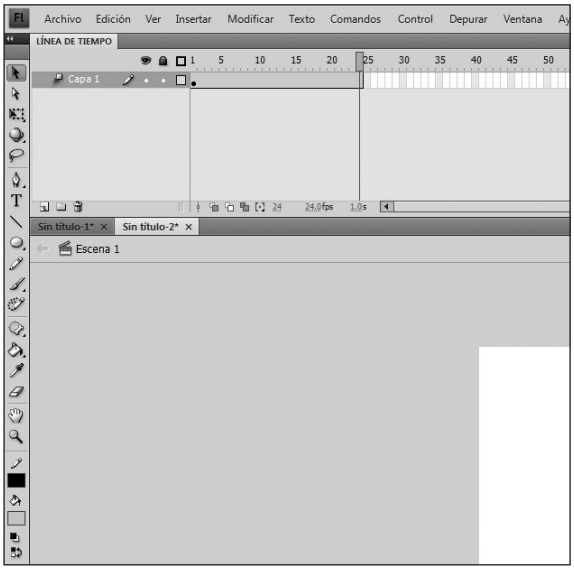


***Figura 5.** La cantidad de frames que ocupa la interpolación la podemos modificar arrastrando desde el final de nuestra animación.*

Luego, nos ubicamos en el último frame de la interpolación y movemos el objeto hasta la posición donde queremos que se realice la animación. Al hacerlo, veremos que se agrega un **keyframe** al final de la interpolación, y así de sencillo es crear una animación. No hay nada nuevo por ahora. Lo que realmente lo hace interesante es la posibilidad de **editar los nodos** que conforman la animación. Vemos que entre el principio y el fin se genera una línea que indica la dirección de la animación. Presionando sobre esa línea, podemos cambiar de posición los nodos y así modificar el **recorrido** de la animación. En versiones anteriores de Flash, lograr esto de forma prolija era sumamente complicado y difícilmente se obtenían los resultados deseados. Ahora, no sólo podemos modificar los puntos que conforman nuestra **guía de movimiento** sino que, modificando la posición o la forma de nuestro objeto en la línea de tiempo, veremos que automáticamente se genera un **keyframe** y nuestra guía de movimiento se adapta a la nueva posición del clip. Animar resulta mucho más completo y fácil que en las versiones anteriores de Flash.

FPS POR DEFECTO

A diferencia de las versiones anteriores, donde la cantidad de **FPS** (**Frames Per Second**, fotogramas por segundo en español) era de 12, en Flash CS4 el valor por defecto es de 24. Si bien se considera que está entre 12 y 24 la cantidad de FPS necesarios para que el ojo humano perciba la sucesión de imágenes como movimiento, 24 FPS aportan mucha más fluidez a nuestras películas.

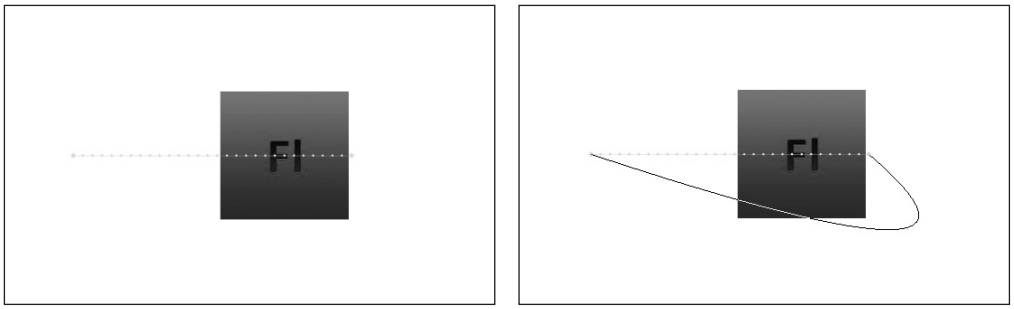


Figura 6. *Cuando creamos una interpolación de movimiento en Flash CS4, podemos editar los nodos de las animaciones y generar el recorrido que deseemos.*

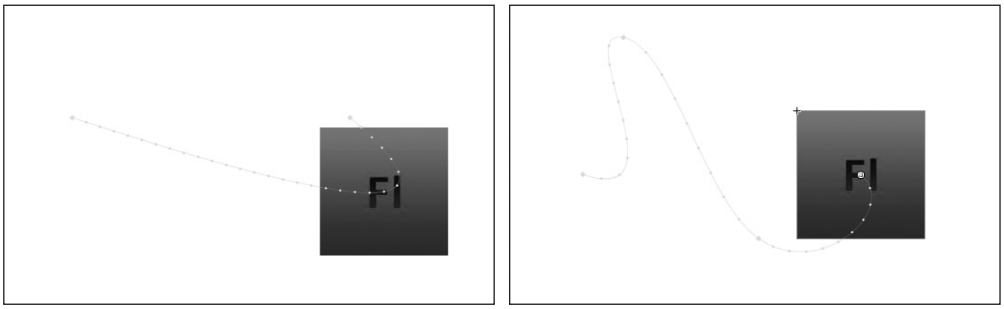


Figura 7. *Cada modificación que hagamos sobre nuestro objeto en su línea de tiempo afecta a la animación en su totalidad.*

Soporte para 3D

En la IDE de Flash CS4 tenemos una nueva herramienta, que nos permite hacer uso del **eje Z** sobre nuestros objetos, es decir, utilizar las **tres dimensiones** para animar.

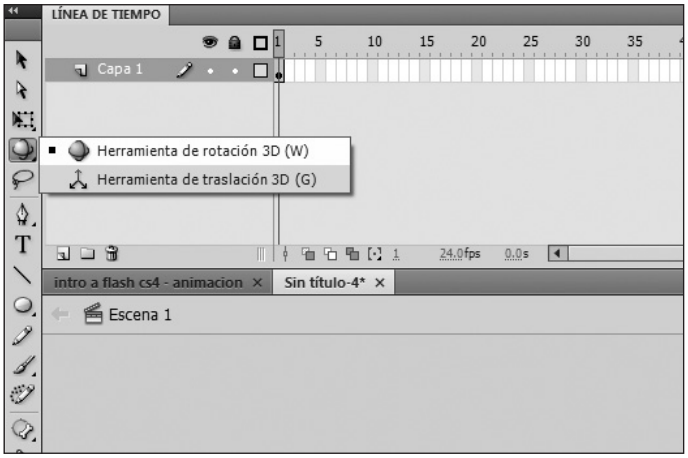


Figura 8. *En la barra de herramientas, encontramos la nueva utilidad de* ***rotación 3D****.*

Si bien este es un paso importante en lo que hace a la incorporación de 3D en Flash, aún queda mucho camino por recorrer. Como podemos ver al probar esta utilidad, las opciones, al menos por ahora, son más bien limitadas.

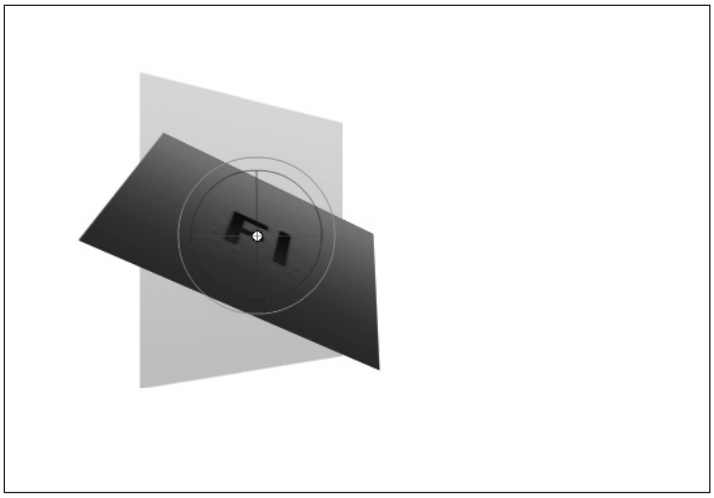


Figura 9. *Ejemplo de modificaciones sobre el* ***eje Z****. El uso de la herramienta es muy sencillo e intuitivo.*

Edición de contenidos y de animaciones

En las versiones anteriores de Flash, si animábamos en **timeline** existían varias limitaciones a la hora de hacer modificaciones sobre una animación ya hecha. Modificar la duración de la animación, la posición y su tamaño o la ubicación de los **MovieClips** implicaba hacer tantos cambios como keyframes tuviésemos, e incluso era prácticamente imposible obtener los mismos resultados. Por su parte, en Adobe Flash CS4, podemos manipular objetos y animaciones por separado. Si trabajamos de forma permanente con animación en timeline, realmente vamos a apreciar estas nuevas características. Presionando sobre la guía de la animación y pulsando la tecla **Q**, podemos editar el tamaño y la forma de la animación, independientemente del MovieClip que estemos empleando en ella.

FLASH 3D: PAPERVISION 3.0 Y SWIFT 3D

Si bien Flash CS4 incorpora una herramienta en 3D para objetos, hace ya algún tiempo que se sigue avanzando en el desarrollo de **Papervision 3.0**, aplicación para implementar 3D en Flash. Podemos encontrar más información al respecto en **http://blog.papervision3d.org**. Otra alternativa para el manejo de 3D en Flash es la utilización de **Swift 3D**. Su sitio web oficial es **www.erain.com**.

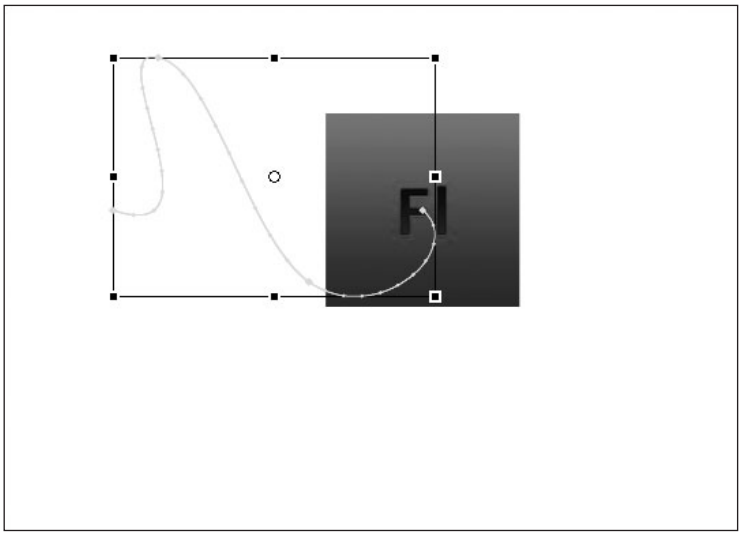


Figura 10. *Podemos modificar el tamaño y la forma de la animación.*

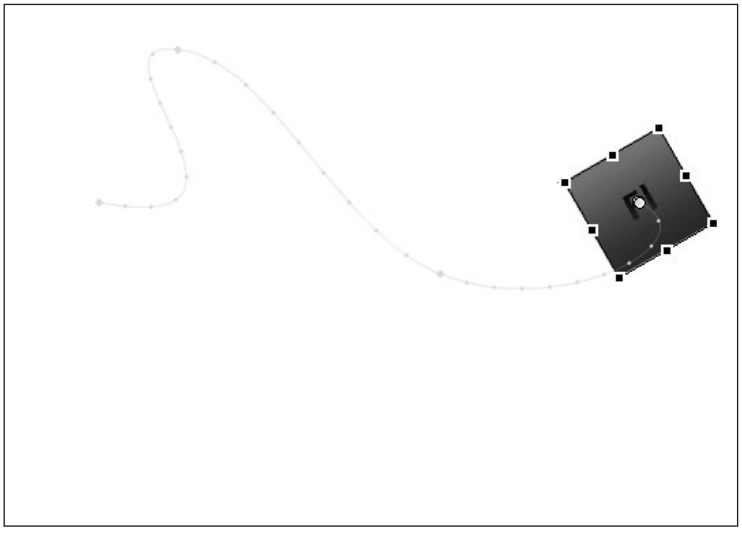


Figura 11. *También es posible editar el MovieClip por separado, sin interferir sobre la animación.*

Otra de las inmensas ventajas que nos proporciona Flash CS4 para nuestras animaciones es la de poder manipularla como un todo. En las versiones anteriores del programa, si nuestra animación ocupaba 20 frames y teníamos 4 keyframes en ella, si optábamos por dilatar o acortar su duración en el tiempo debíamos volver a posicionar cada uno de los objetos que había sobre cada keyframe. Ahora, al escalar la animación, ésta respeta la distribución de los objetos y podemos modificar su duración en el tiempo haciendo un **drag** con ella, es decir, arrastrando sus frames según lo que necesitemos (estirar o acortar).

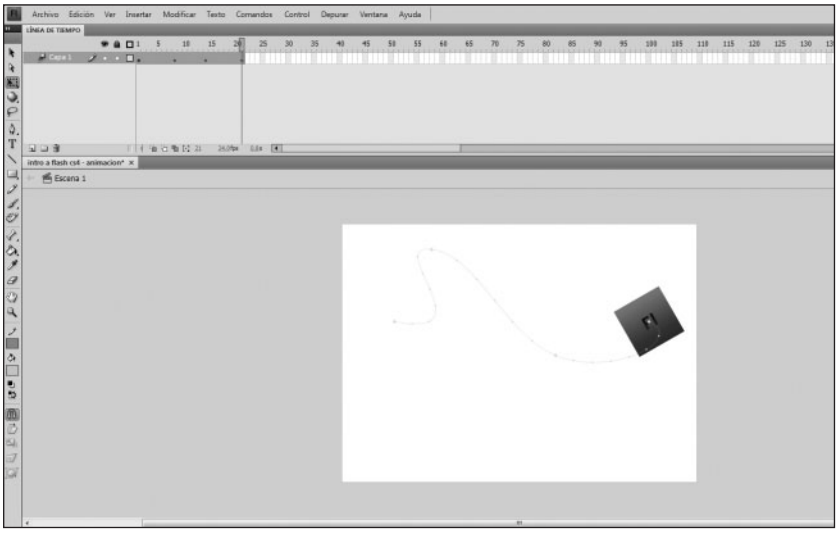


Figura 12. *Al modificar la cantidad de frames de nuestra animación, los keyframes respetan su equidistancia del resto.*

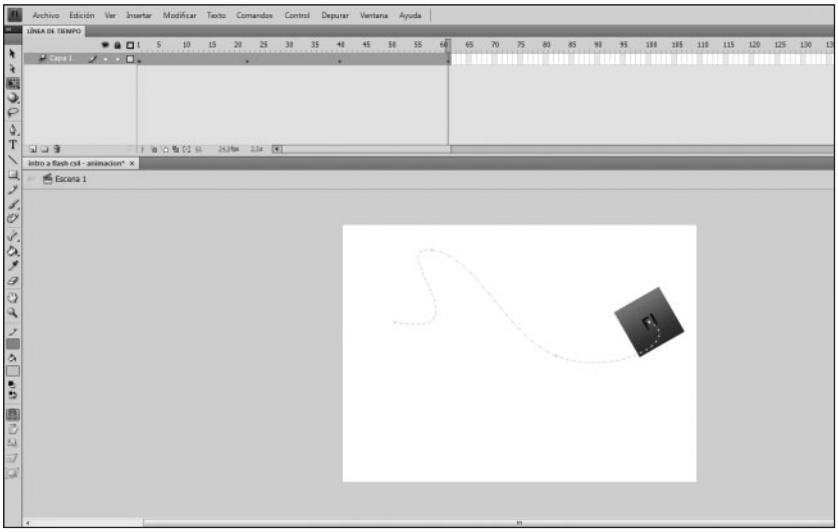


Figura 13. *Como vemos, los keyframes están más distantes.*

Editor de movimiento

Flash CS4 cuenta con un completo **EDITOR DE MOVIMIENTO**. Desde él podemos controlar de manera detallada cada una de las transformaciones a las que sometemos a nuestros objetos. Dentro de esta herramienta, contamos con una serie de paneles en los cuales se encuentran agrupadas las distintas propiedades que es posible editar: dentro del panel **Movimiento básico** podemos modificar la posición sobre los ejes **X**, **Y** y **Z**, dentro de **Transformación** contamos con las **escalas** para **x** e **y**.

A su vez, también podemos agregar efectos en la sección **Efectos de color** (**alfa**, **brillo**, **tinta**, **color avanzado**) y desde **Suavizados**. Como veremos, el panel nos proporciona un control mucho más exacto y estricto sobre cada una de las propiedades a las que les aplicamos transformaciones.

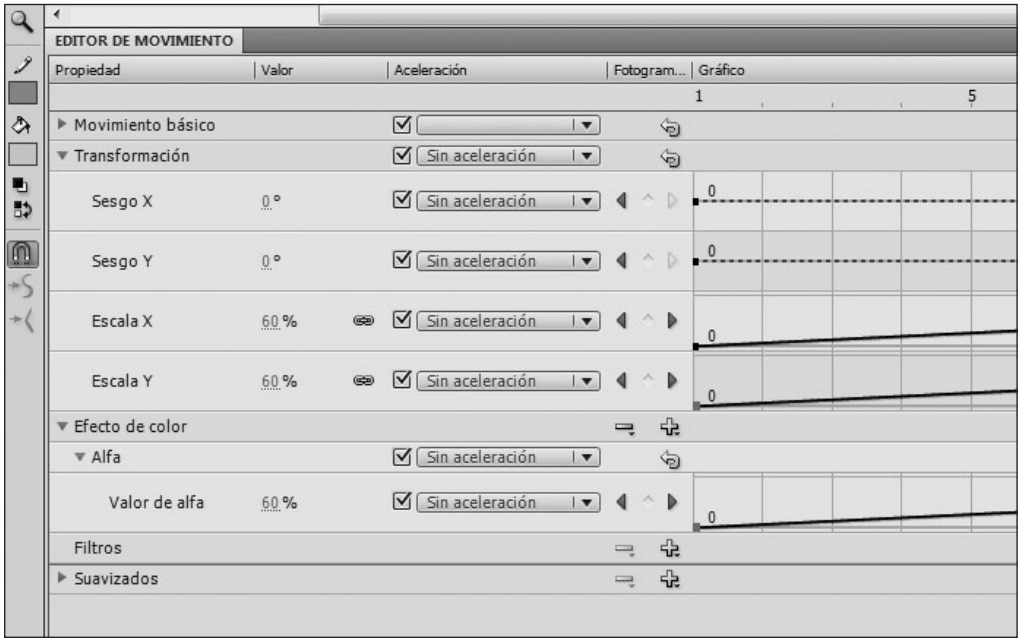


Figura 14. *Si bien puede resultar confuso al principio, una vez que nos acostumbramos a usarlo, el* ***EDITOR DE MOVIMIENTO*** *se convierte en una herramienta indispensable para la creación de animaciones.*

Configuración predefinida de movimientos

Flash CS4 cuenta con una **librería** que contiene una serie de animaciones listas para ser utilizadas. Para visualizar el panel **Configuración predefinida de movimientos**, debemos ir a **Ventana** y luego seleccionar ese ítem de la lista, o bien desde el menú **CONCEPTOS BÁSICOS** marcar la opción **ANIMADOR**, que hará aparecer este menú en

FLASH Y SU NUEVA FAMILIA: LA COMPAÑÍA ADOBE

El software Flash perteneció a la compañía Macromedia hasta 2005, año en el cual fue comprada por la empresa Adobe. Desde ese entonces, Adobe tuvo como meta integrar y potenciar las prestaciones de sus productos con los de Macromedia. Esta unión fue de gran importancia para la implementación y fusión de las herramientas de ambas empresas.

el borde inferior izquierdo de nuestra interfaz. Para elegir una de las animaciones disponibles, debemos hacer clic con el botón derecho sobre la que deseemos utilizar y luego seleccionar **Aplicar en posición actual**.

Figura 15. *Desde el panel* ***CONFIGURACIÓN PREDEFINIDA DE MOVIMIENTOS*** *podemos importar directamente las animaciones al escenario. Dentro del mismo panel es posible guardar nuestras propias animaciones.*

Huesos

Los huesos son otra de las nuevas herramientas con las que contamos en la IDE de Adobe Flash CS4. Por medio de este método (**cinemática inversa**), podemos animar objetos individuales o conjuntos de objetos que se encuentran entrelazados entre sí por una **estructura de huesos**. De este modo, al mover un objeto, vamos a modificar la posición de todos aquellos elementos que estén vinculados a aquél que estamos variando por medio de esta estructura.

AYUDA ONLINE

En **http://help.adobe.com/es_ES/Flash/10.0_UsingFlash/** encontraremos la documentación oficial de **Adobe Flash** en español. Esta información nos resultará de mucha utilidad en caso de tener dudas o consultas. También hallaremos actualizaciones periódicas, ayuda en línea y la opinión de usuarios y expertos en la materia, exponiendo problemas y soluciones.

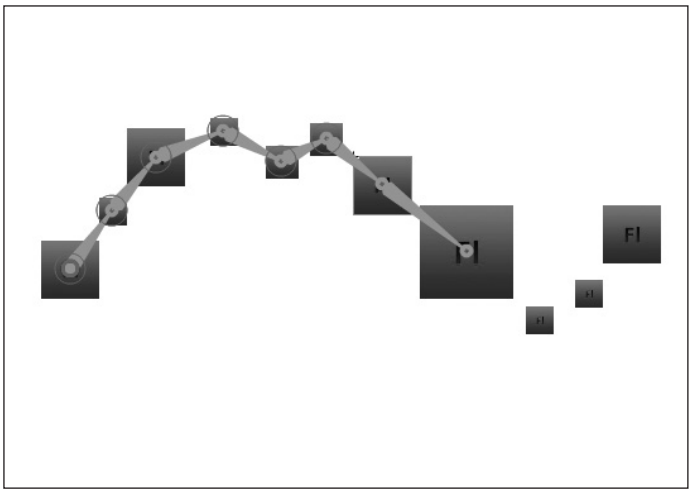


Figura 16. *Para crear estructuras de huesos debemos indicar el punto de inicio y de finalización.*

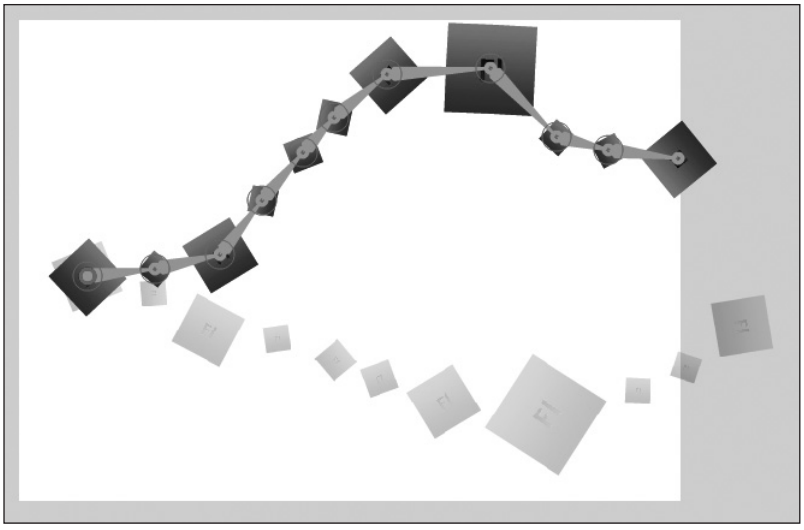


Figura 17. *Podemos asignar estructuras de huesos solamente a MovieClips.*

Panel Proyecto

Para visualizar el panel **PROYECTO** debemos dirigirnos a **Ventana/Otros paneles** y allí elegir **Proyecto** dentro de la lista de opciones, o bien desde el menú **CONCEPTOS BÁSICOS**, al seleccionar la opción **DESARROLLADOR** este menú aparecerá en el borde superior izquierdo. Si bien este panel ya existía con anterioridad, no se lo utilizaba por su poca practicidad. En la versión actual, se adapta de mejor manera a las características de un desarrollo, permitiéndonos un mayor control sobre los archivos que se ven implicados en un desarrollo en particular.

Figura 18. *El panel PROYECTO nos permite tener un mejor control y manejo sobre los archivos que conforman el proyecto que estamos desarrollando.*

Haciendo clic en **Proyectos** y luego en **Nuevo proyecto...**, Flash nos pedirá que indiquemos el nombre y el directorio dentro del cual se encuentra. Una vez creado el proyecto, si el directorio tiene contenidos, veremos la estructura de archivos y carpetas.

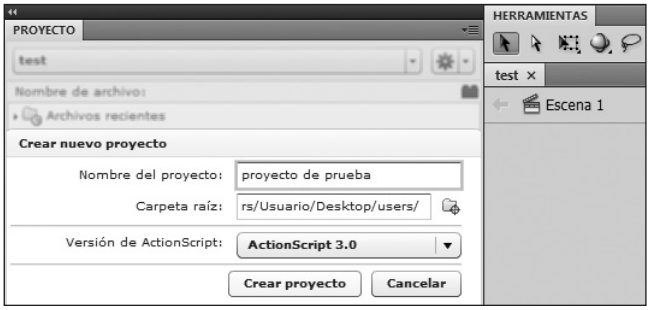


Figura 19. *Desde el panel definimos el nombre de nuestro proyecto y el directorio en el que se encuentra.*

Lo que resulta muy útil de este administrador de proyectos es que podemos crear nuestras clases desde aquí, y su estructura se generará automáticamente, evitándonos la necesidad de hacerlo manualmente. El concepto de clases y la programación orientada a objetos son cuestiones relativamente nuevas en el entorno de desarrollo Flash. En el próximo capítulo profundizaremos sobre estos temas, por lo que si no dominamos el lenguaje o no contamos con conocimientos de programación, no debemos alarmarnos al no comprender estos conceptos, que exponemos aquí de forma introductoria.

Publicación para ADOBE AIR

En esta nueva versión de Flash podemos exportar directamente para **Adobe AIR**. También es posible, al abrir el programa, optar por seleccionar la creación de un archivo de Flash para Adobe AIR. Todo esto facilita y fomenta, en gran medida, el desarrollo de

aplicaciones bajo esta nueva tecnología. Si bien veremos más al respecto de Adobe AIR en un capítulo dedicado íntegramente a su uso, vale adelantar que se trata de un entorno multiplataforma para el desarrollo de **aplicaciones RIA** de escritorio. Estas aplicaciones se pueden desarrollar en **Flex**, **Flash**, **HTML** o **AJAX**. En **www.adobe.com/es/products/air/** encontramos información respecto a la plataforma, sus características, alcances y proyectos realizados en AIR, entre otras cosas.

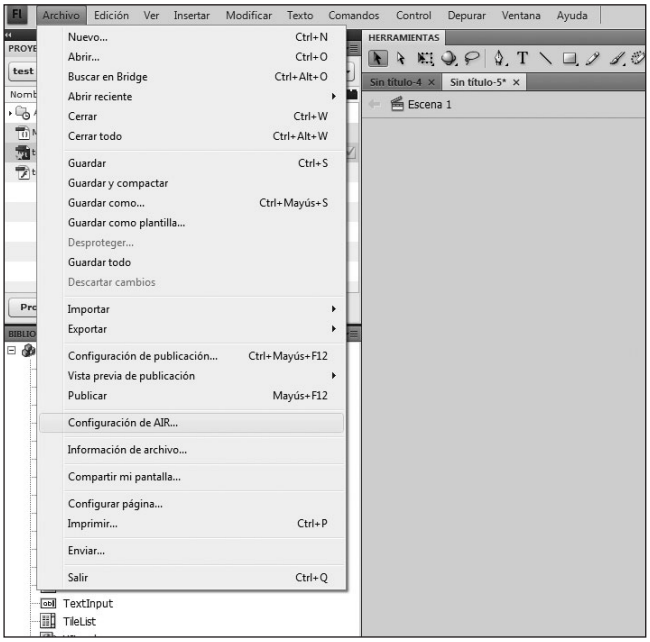


Figura 20. *Desde **Archivo/Configuración de AIR...** podemos determinar todas las opciones para la exportación de nuestro archivo.*

... RESUMEN

Así como la aparición de la versión Flash CS3 representó un cambio trascendental al migrar a ActionScript 3.0, en Adobe Flash CS4 es evidente que las modificaciones más importantes se dieron en los campos de la animación y el diseño. Esta primera parte tuvo la intención de dar a conocer las nuevas características del programa. Una vez incorporadas, nos adentraremos en lo que verdaderamente nos permitirá llevar a cabo los desarrollos que iremos realizando a lo largo del libro: el manejo de ActionScript 3.0.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1) ¿En qué área se produjo la mayor cantidad de mejoras en Flash CS4?

2) ¿Desde qué menú podemos cambiar la distribución de los contenidos de la interfaz para adaptarlos a nuestras necesidades?

3) ¿Qué opciones recuerda dentro de este menú?

4) ¿Qué herramientas se incorporaron en esta versión para facilitar la manipulación de objetos sobre el eje Z?

5) ¿Cuál es la función del panel Editor de Movimiento?

6) ¿Qué propiedades nos permiten modificar este panel?

7) ¿Qué herramienta nos posibilita unir varios objetos individuales o conjuntos de objetos a fin de vincularlos y manipularlos como estructuras?

8) ¿Cuál es la principal ventaja de vincular objetos?

9) ¿Cuál es la función del panel Proyecto?

10) ¿Cuál es su principal ventaja?

EJERCICIOS PRÁCTICOS

1) Si cuenta con Flash CS3, analice las principales diferencias que hay en la creación de animaciones, haciendo foco en las ventajas que incorpora Flash CS4 respecto a este tema.

2) Realice diversas animaciones implementando el nuevo Editor de movimientos. Familiarizarse con él es fundamental para la animación en timeline.

3) Genere animaciones en 3D haciendo uso de la nueva herramienta con la que cuenta Flash CS4.

4) Emplee la herramienta Huesos para la creación de un personaje. Genere sus articulaciones por medio de esta utilidad.

5) Cree un objeto y aplíquele diversas configuraciones predefinidas de movimientos. Analice las características de los distintos tipos de animaciones y para qué casos podrían ser de utilidad.

ActionScript 3.0

A pesar de que ActionScript 3.0 tiene una gran cantidad de similitudes en lo que a sintaxis del lenguaje se refiere en relación a sus versiones anteriores (ActionScript 1.0 y ActionScript 2.0), probablemente sea mejor pensarlo como un lenguaje nuevo. ActionScript 3.0 nos provee de una sintaxis completamente mejorada, con cambios de gran importancia en cuestiones de desarrollo.

NUEVOS CONCEPTOS Y MEJORAS DE ACTIONSCRIPT 3.0

Probablemente, lo que atemoriza a gran parte de la comunidad de desarrolladores es la gran brecha entre la modalidad de desarrollo que se utilizaba en las versiones previas del lenguaje y la que se implementa en la versión actual. Nuestra intención, por medio de esta obra, es suplir esa distancia, y el modo más apropiado de encarar este salto es pensando que todas las modificaciones surgidas son en pos de la enorme cantidad de mejoras que trae esta última versión, haciendo de **ActionScript 3.0** un lenguaje verdaderamente robusto sobre el que podemos tener un control más eficiente de nuestros desarrollos.

Las principales mejoras de ActionScript 3

ActionScript 3.0 pone a nuestra disposición una serie de mejoras en relación a sus versiones anteriores, que vale la pena que veamos. Si bien estos son los aspectos fundamentales en los que vamos a notar grandes cambios, existen progresos prácticamente en todas las clases que abarca ActionScript 3.0. Muchas de ellas las iremos viendo a lo largo del desarrollo de este libro.

- **Control de errores en tiempo de ejecución**: en caso de existir errores en nuestro código, por medio del **debug** se nos informa sobre ellos a través de un código de error y una descripción que nos ayudadarán a resolverlos.
- **Nueva estructura de objetos**: agregar, quitar, modificar la posición de los objetos en el entorno de desarrollo es sencillo en comparación con las versiones anteriores del lenguaje. Esto nos permite manejarnos bajo un entorno mucho más claro y prolijo.
- **Nueva estructura de eventos**: probablemente, ésta sea una de las mejoras de mayor importancia. Los eventos, ya sean de mouse, teclado u otros, se ejecutan y controlan por medio de **event listeners**. Esto nos permite un manejo mucho más preciso y potente sobre lo que sucede en nuestra película, logrando optimizar enormemente nuestros desarrollos.

AYUDA ONLINE

En el sitio **http://help.adobe.com/es_ES/ActionScript/3.0_ProgrammingAS3/** encontramos prácticamente toda la documentación en español del lenguaje. Puede resultarnos de gran utilidad si deseamos interiorizarnos en cuestiones muy puntuales del lenguaje o como guía de consulta. Es posible descargar un **.pdf** del sitio con todos los contenidos.

- **Mejoras en el manejo de estructuras XML**: gracias a la sintaxis **E4X** (**ECMAScript for XML**), manejar estructuras XML en ActionScript 3.0 es mucho más sencillo que en sus versiones anteriores. Esto hace más simple la navegación dentro de estructuras y su manipulación.
- **Mejoras en el manejo de sonidos**: ActionScript 3.0 permite el manejo de sonidos asignando canales separados a cada uno de ellos. Otra de las grandes novedades es la posibilidad de acceder a la amplitud y frecuencia de los espectros de sonido, al igual que a la intensidad del sonido de cada parlante, pudiendo crear representaciones gráficas reales para nuestras reproducciones.
- **Mejoras en la Programación Orientada a Objetos**: el modo de programación orientada a objetos fue mejorado en ActionScript 3.0, haciendo más legible la sintaxis y ofreciendo un control mucho más estructurado sobre nuestros códigos.

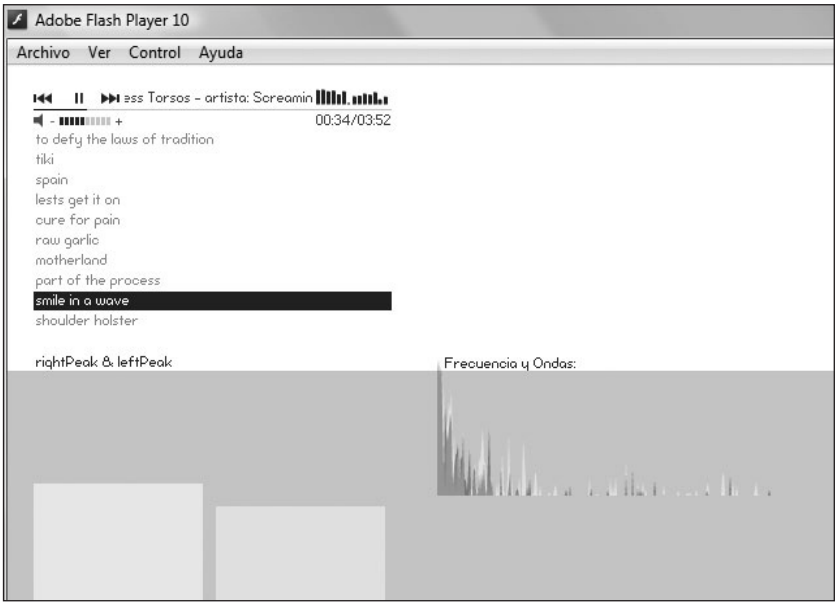


Figura 1. *Dibujo de espectros de sonido por medio del método* ***computeSpectrum()*** *de la clase* ***SoundMixer****; una de las tantas mejores de AS3 que veremos en este libro.*

MIGRAR A AS3

En este apartado veremos los conceptos básicos para comenzar a programar en ActionScript 3.0. A no desesperar: la **sintaxis** de AS3 no varía de forma drástica respecto a sus versiones antecesoras, y no debemos olvidar que todos los cambios tienen como meta solucionar algunas falencias de la plataforma y hacer de ActionScript 3.0 un lenguaje de programación confiable.

Variables y tipos de variables

ActionScript 3.0 es más **estricto** con el uso de las variables y esto es un beneficio enorme para los desarrolladores, ya que de esta manera nos evitamos una gran cantidad de problemas. En las versiones anteriores de AS no era necesario el uso de la sentencia **var** al declarar una variable. A partir de esta nueva versión, sí lo es. El modo correcto de declarar una variable es el siguiente: **var miVariable;**.

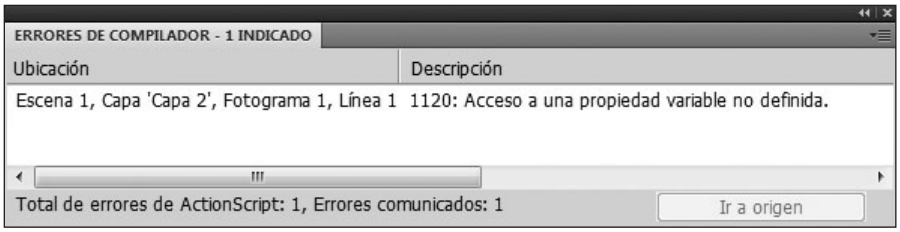


***Figura 2.** En caso de omitir la sentencia var al declarar la variable, Flash emitirá una alerta respecto del error.*

Tipos de variables

Al declarar una variable, es recomendable asociarla con un determinado **tipo de dato**. Para asignar un tipo de dato, debemos hacerlo del siguiente modo: **var miVariable:String;**. Una de las ventajas de definir a qué tipo de dato corresponde una variable es la de evitar operaciones incorrectas con ellas y, además, desde el momento en que la declaramos, Flash sabe qué operaciones son válidas y cuáles no para ella.

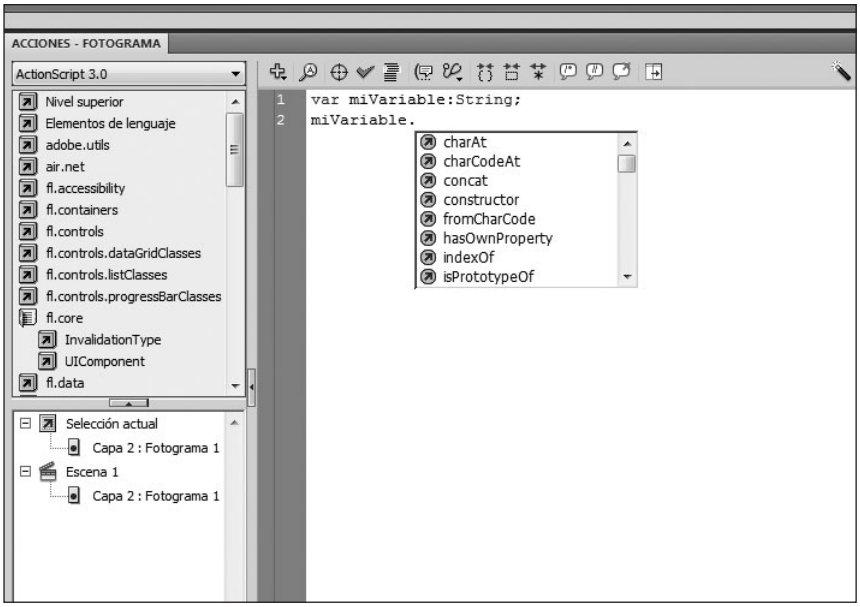


***Figura 3.** Al colocar un punto al final de una variable, se desplegará un menú con las propiedades y métodos disponibles para ella.*

El valor de una variable podemos asignarlo al declararla:

```
var miVariable:String = “esta es mi variable del tipo string”;
```

O bien después:

```
var miVariable:String;
miVariable = “esta es mi variable del tipo string”;
```

Otro detalle que debemos tener en cuenta es que si declaramos una variable y no su valor, hay determinados tipos de variables que ya vienen con un valor predeterminado, como podemos ver en la **Tabla 1**.

TIPO DE DATOS	VALOR PREDETERMINADO
Boolean	false
Int	0
Number	NaN
Object	null
String	null
uint	0

***Tabla 1.** Tipos de variables con sus valores predeterminados.*

Propiedades

La mayoría de los objetos que manipulamos en Flash tiene propiedades a las cuales podemos acceder. En ciertos casos, es posible leerlas o modificarlas y en otros solamente leerlas. Hay algunos cambios respecto a las versiones anteriores de Flash. A continuación, veremos cuáles son:

- El **rango de valores** que solía ser de 0 a 100, en AS3 es de 0 a 1. En **ActionScript 2.0**, para indicar un valor **alpha** de un 50% a un **MovieClip**, debíamos hacerlo de este modo: **miClip._alpha = 50;**. En ActionScript 3.0, es **miClip.alpha = 0.5;**, o bien **miClip.alpha = .5;**.
- **Las propiedades ya no comienzan con guión bajo**. Para modificar el ancho (**width**) y el largo (**height**) de un **MovieClip**, debíamos hacerlo de la siguiente manera:

```
miClip._width = 50;
miClip._height = 50;
```

En ActionScript 3.0, la sintaxis de las propiedades de los objetos no lleva este guión:

```
miClip.width = 0.5;
miClip.height = 0.5;
```

- Por último, es importante mencionar las propiedades **_xscale** y **_yscale**, que han sido reemplazadas en AS3 por **scaleX** y **scaleY**.

DisplayList: los elementos visuales en ActionScript 3.0

Este es, sin dudas, uno de los mayores cambios que vamos a encontrar en este proceso de migración a AS3. La manera en la que manejamos los elementos visuales de nuestro entorno es completamente distinta a la que empleábamos en las versiones anteriores del lenguaje. En esas versiones, era relativamente complejo y desprolijo el manejo de nuestros elementos visuales, no todos requerían de la misma sintaxis y el modo en el que llevábamos a cabo nuestros desarrollos influenciaba en las posibilidades con las que luego íbamos a contar. En AS3, tenemos una **única sintaxis** para el manejo de nuestros elementos visuales, los cuales se encuentran en lo que a partir de aquí llamaremos **DisplayList** (también la denominaremos **lista de visualización**). Veremos que su uso es verdaderamente sencillo e intuitivo, y a la vez nos permite el manejo de objetos con los que no contábamos anteriormente y nos van a ser de gran utilidad.

Estructura

La mejor manera de pensar el **DisplayList** es como un **contenedor** dentro del que se encuentran todos los objetos. Su primer elemento es el **escenario** (**stage**), que es el contenedor principal de nuestros desarrollos y sobre el cual disponemos nuestros componentes. De forma inmediata en esta jerarquía, tenemos la línea de tiempo (**timeline**), que es el elemento esencial de cualquier película. Si bien el escenario es el contenedor para todos nuestros objetos, la línea de tiempo nos permite ubicarlos sobre sus fotogramas y hacer manejo del tiempo. Estos dos primeros elementos no son creados por nosotros sino por el propio **player** al comenzar un nuevo proyecto.

INFORMACIÓN RESPECTO A RIAS

En **www.adobe.com/resources/business/rich_internet_apps/#open** encontraremos información interesante respecto a la creación de **Rich Internet Applications** (**RIAS**) y a las tecnologías que se emplean en cada caso. Se toman como ejemplos las más diversas empresas que llevaron a cabo desarrollos de este tipo, con información sobre Flex, Adobe Air, Ajax y Flash.

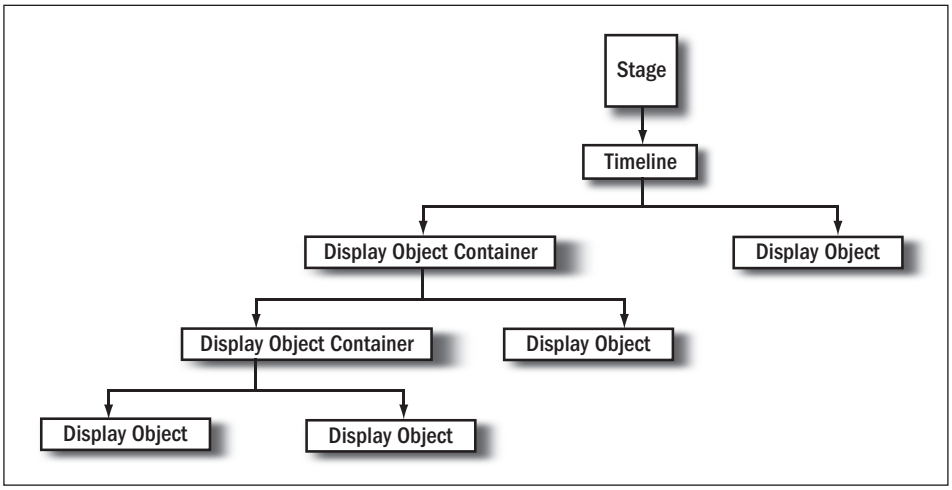


Figura 4. *El escenario y la línea de tiempo son creados por* ***Flash Player*** *y sobre ellos disponemos los contenidos.*

Por último se encuentran todos los elementos visuales de nuestro desarrollo. Y aquí también tenemos una cuestión de importancia para entender correctamente el funcionamiento de la **DisplayList** o **lista de visualización**: si bien hay elementos visuales finales, también hay otros cuya función es la de ser contenedores para otros elementos (**containers**). A estos dos tipos de elementos los vamos a diferenciar en **Display Objects** y **Display Objects Containers**. Entender este concepto y dominarlo con claridad nos va a ser sumamente útil cuando encaremos proyectos de envergadura. Este uso jerárquico nos permite ser mucho más ordenados y tener un mayor control sobre nuestras estructuras.

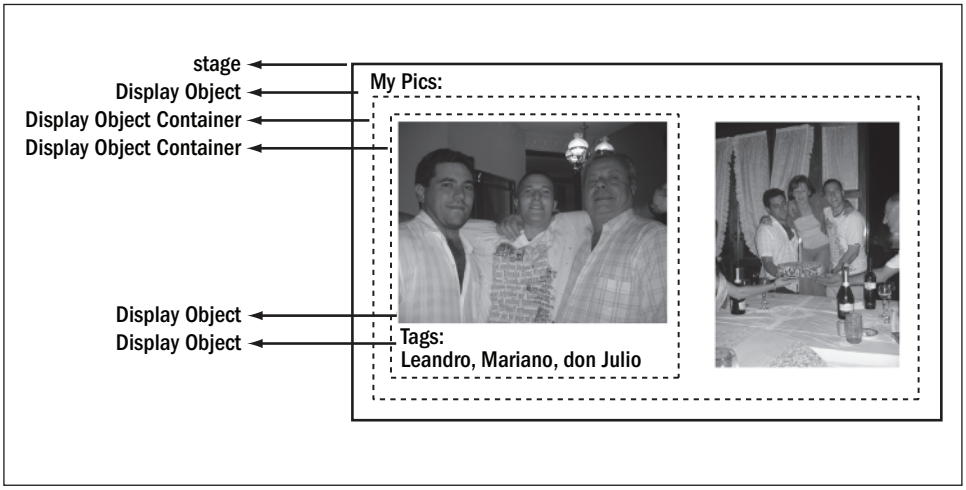


Figura 5. *Al comprender esta imagen, entenderemos la diferencia entre un* ***Display Object*** *y un* ***Display Object Container.***

Crear instancias: operador new

En ActionScript 3.0, los elementos con los que contamos para llevar a cabo nuestros desarrollos son clases: la clase **MovieClip**, la clase **Sprite**, etcétera. Entonces, cuando decimos que contamos con un MovieClip, en realidad tenemos una instancia de la clase **MovieClip**. Para crear una instancia de una clase debemos utilizar el operador **new**. Para comprenderlo, veamos un ejemplo. La siguiente sentencia crea una instancia de la clase **MovieClip**, que lleva el nombre **miClip**:

```
var miClip:MovieClip = new MovieClip();
```

Una vez creada una instancia, podemos acceder a las propiedades y métodos de la clase sobre la que creamos la instancia:

```
var miClip:MovieClip = new MovieClip();
miClip.alpha = 0.4;
```

Agregar objetos dinámicamente: addChild();

Su uso es sumamente sencillo y, a diferencia de ActionScript 1.0 y ActionScript 2.0, su sintaxis es más fácil de implementar. Anteriormente, para agregar objetos dinámicamente al escenario o dentro de otros contenedores, utilizábamos los métodos **duplicateMovieClip()** y **attachMovie()**. Estos métodos no siguen en vigencia, ya que en ActionScript 3.0, para agregar un elemento de visualización, simplemente tenemos que crear una instancia por medio de su constructor:

```
var miClip:MovieClip = new MovieClip();
```

Y luego debemos utilizar el método **addChild()** para incorporarlo, ya sea al escenario (**stage**) o dentro de algún contenedor, como vemos a continuación:

SIMPLEZA EN EL MANEJO DE CONTENIDOS

Si estamos migrando a **AS3** desde **AS2**, valoraremos la manera sencilla de manejar los contenidos visuales. En AS2, primero teníamos que crear un objeto y luego duplicarlo (**duplicateMovieClip**) o hacer un **attach** (**attachMovie**) de él. Estos procedimientos ya no se usan en ActionScript 3.0, donde generamos el nombre de la instancia y la agregamos al escenario o a su contenedor.

```
addChild(miClip);
```

Luego, el código que vemos a continuación nos permitirá ver en detalle el funcionamiento del método **addChild()**:

```
var myContainer:Sprite;
var myChildren:Sprite;

myContainer = new Sprite();
myContainer.x = stage.stageWidth/2 - 25;
myContainer.y = 25;

addChild(myContainer);

for(var m:uint = 0; m<20; m++){
     myChildren = new Sprite();
     myChildren.y = 11 * m;
     myChildren.graphics.beginFill(0xff0066, 0.5);
     myChildren.graphics.drawRect(0, 0, 50, 10);
     myChildren.graphics.endFill();
     myContainer.addChild(myChildren);
}
```

En primer lugar, centramos **sprite** contenedor, que lleva el nombre **myContainer**, y lo situamos en el medio del escenario. Luego, por medio de un bucle **for**, agregamos 20 sprites generados dinámicamente en su interior. Es importante que notemos de qué manera utilizamos el método **addChild()**. En primer lugar, lo usamos para añadir al escenario nuestro contenedor principal (**display object container**) y, luego, lo utilizamos dentro del bucle **for** para añadir los sprites (**display objects**) dentro del **display object container** (**myContainer**).

AGREGAR ELEMENTOS DE LA LIBRERÍA

No debemos confundir el concepto de crear contenidos dinámicamente por medio de código, tales como **Sprites** o **MovieClips**, con incluir clips dinámicamente desde la librería en el escenario. Ese procedimiento en cuestión lo veremos en las próximas páginas.

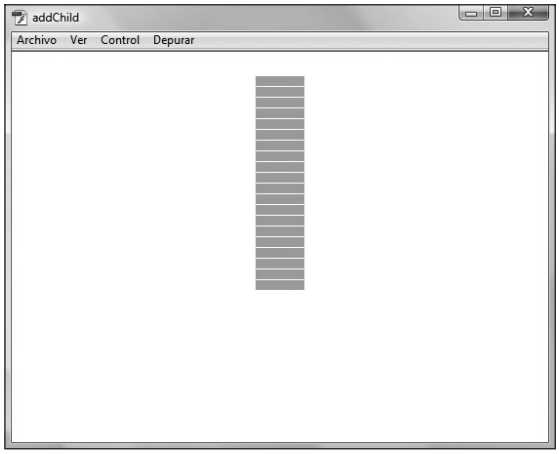


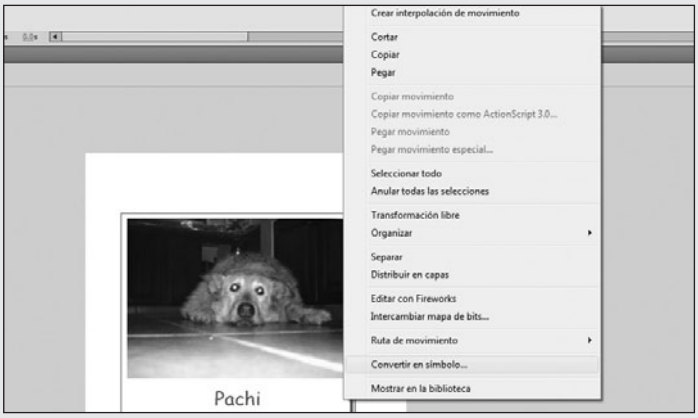
Figura 6. *El método* ***addChild()*** *en funcionamiento: los sprites generados dinámicamente se ubicarán dentro del sprite* ***myContainer****.*

Agregar símbolos de la biblioteca al escenario

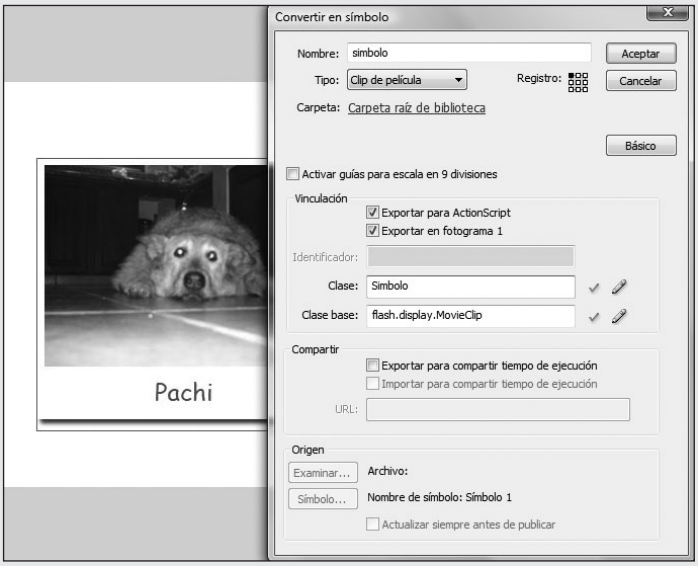
En las versiones previas del lenguaje, para manipular un elemento de nuestra biblioteca por medio de código, debíamos vincularlo y crear una instancia de él. Como vimos anteriormente, en ActionScript 3.0, esto lo hacemos por medio de **clases**. Lo que instanciamos son clases, no objetos. Para agregar un elemento de la biblioteca al escenario lo hacemos de la siguiente manera:

■ Agregar un símbolo al escenario PASO A PASO

1 Una vez dentro de Flash, importe un elemento al escenario. Selecciónelo, presione el botón derecho del mouse y elija la opción **Convertir en símbolo...** (o presione la tecla **F8**, el atajo de teclado).

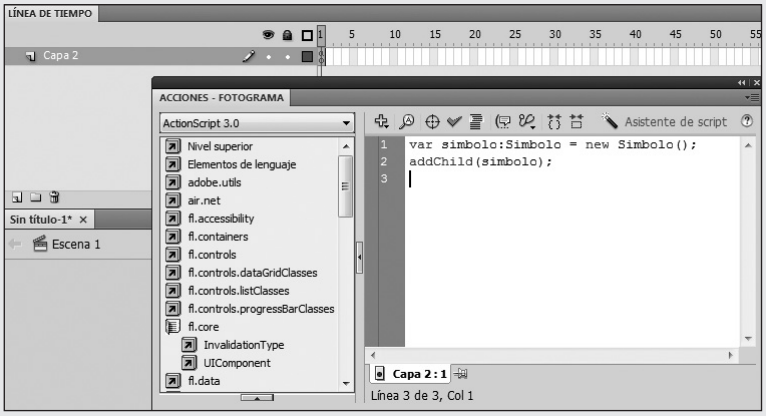


2 En la ventana **Convertir en símbolo**, defina el nombre que tendrá el elemento dentro de la biblioteca. Marque la casilla **Exportar para ActionScript** en la sección **Vinculación** y defina el nombre de la clase. En este caso, se denominó **simbolo** a la instancia y **Simbolo** a la clase.

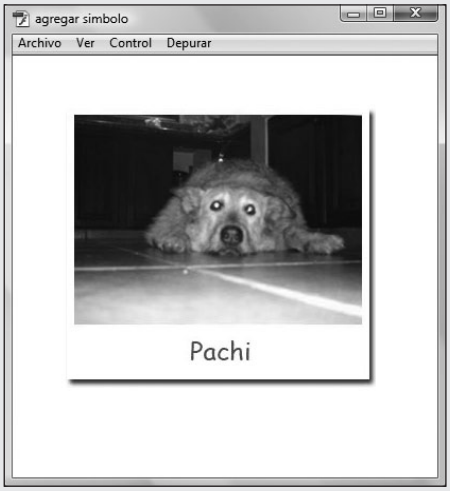


3 Quite el elemento del escenario y posiciónese sobre el primer frame de la línea de tiempo. Haga clic con el botón derecho y seleccione **Acciones** o presione **F9**, atajo del panel. Allí, ingrese el siguiente código:

```
var simbolo:Simbolo = new Simbolo();
addChild(simbolo);
```



4 Para terminar, exporte la película presionando la combinación **Control + Enter**.



Una vez que hemos creado el símbolo, al haberlo asignado como una clase (en nuestro caso de un MovieClip), éste va a heredar todos sus comportamientos. Podemos probar rotándolo, modificando su tamaño y su posición:

```
var simbolo:Simbolo = new Simbolo();
simbolo.x = simbolo.y = 50;
símbolo.rotation = 10;
addChild(simbolo);
```

En algunas ocasiones, vamos a necesitar crear nuestros objetos dinámicamente y en otras los vamos a tener que vincular desde la **biblioteca**. Esto, lógicamente, varía según la complejidad del desarrollo y de lo que queramos hacer. Ningún método es mejor que otro, simplemente son distintos y nos sirven para diferentes cosas.

¿QUÉ ES UN SPRITE?

Como hemos visto, **sprite** es un nuevo objeto con el que cuenta Flash desde el nacimiento de ActionScript 3.0. La mejor manera de entenderlo es como un MovieClip, pero que, a diferencia de éste, no cuenta con su respectiva línea de tiempo. Entonces, para comprender mejor de qué se trata, podemos decir que es un **MovieClip de un único frame**.

AddChildAt(): adiós a getNextHighestDepth()

Anteriormente, en AS2, cuando instanciábamos un MovieClip desde la biblioteca o lo duplicábamos, utilizábamos el método **getNextHighestDepth()**, por medio del cual teníamos que indicar el **nivel** en el que íbamos a colocar nuestro clip. No vamos a volver a necesitar este método ya que **addChild()** se encarga de asignar el nivel de manera automática, agregando siempre el objeto en cuestión al final de la lista de visualización (**DisplayList**). En el caso de que debamos agregar un elemento en otra posición que no sea la última, contamos con el método **addChildAt()** que, a diferencia del método **addChild()**, permite indicar el nivel en el que queremos posicionar nuestro clip dentro de una estructura.

A continuación, analizaremos el código necesario para ver en funcionamiento el método **addChildAt()**. El resultado será una película con dos círculos: al segundo de ellos lo agregaremos por medio del método **addChildAt()** y veremos que no se ubica al final de la lista de visualización ocupando la última posición, sino que al especificarle la posición **0** como parámetro, se ubicará en el primer lugar de la lista de visualización, por debajo del círculo que generamos primero.

```
var myContainer:Sprite;
var myCircle:Sprite;
var mySecondCircle:Sprite;

myContainer = new Sprite;
myContainer.x = stage.stageWidth/2 - 25;
myContainer.y = 150;
addChild(myContainer);

myCircle = new Sprite();
myCircle.graphics.beginFill(0xff0066, 1);
myCircle.graphics.drawCircle(0, 0, 50);
myCircle.graphics.endFill();
myContainer.addChild(myCircle);

mySecondCircle = new Sprite();
mySecondCircle.x = 2;
mySecondCircle.y = 2;
mySecondCircle.graphics.beginFill(0x555555, 1);
mySecondCircle.graphics.drawCircle(0, 0, 50);
mySecondCircle.graphics.endFill();
myContainer.addChildAt(mySecondCircle, 0);
//myContainer.addChild(mySecondCircle);
```

Para entender estos conceptos con mayor claridad, una buena práctica es quitar las dos barras que comentan la última línea de código y comentar la anteúltima, para alternar el funcionamiento de los métodos **addChild()** y **addChildAt()** y notar sus diferencias.

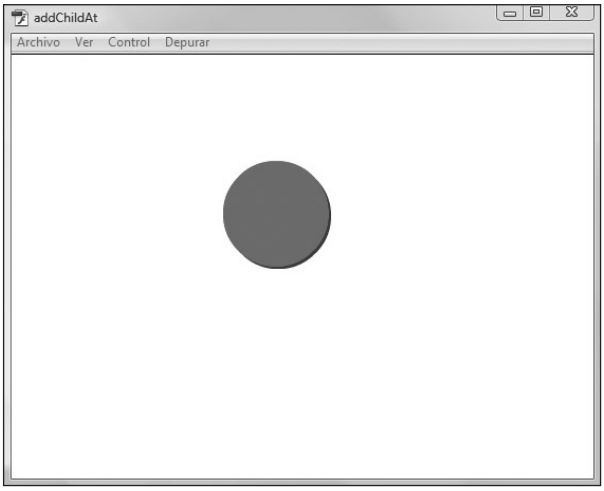


Figura 7. *Dos círculos sobre nuestra película; si queremos asignarle la profundidad al elemento de visualización, debemos utilizar el método addChildAt(posición) en lugar del método addChild().*

Eliminar contenidos: removeChild() y removeChildAt()

En versiones anteriores, utilizábamos el método **removeMovieClip()** para eliminar MovieClips. En ActionScript 3.0, la manera en la que quitamos contenidos posee una sintaxis muy similar a la que empleamos para agregar elementos. Podemos hacerlo por medio de **removeChild()**, indicando como parámetro el objeto que se va a eliminar:

```
var mySprite:Sprite = new Sprite();
addChild(mySprite);
removeChild(mySprite);
```

III ADDCHILD() Y ADDCHILDAT()

Si bien es necesario comprender ambos conceptos y las características de cada uno, a lo largo del libro veremos que, generalmente, resolveremos nuestros códigos empleando **addChild()**. Por su parte, **addChildAt()** es utilizado sólo en algunos casos puntuales donde el uso de niveles es realmente imprescindible para el desarrollo.

También podríamos indicar el nivel en el que queremos sacar un elemento, ingresando la sentencia **myContainer.removeChildAt(0);**. Es importante saber que por medio de la propiedad **numChildren** es posible conocer qué cantidad de elementos contiene un Sprite o MovieClip contenedor. Esto resulta especialmente útil cuando estamos eliminando elementos por medio del método **removeChild()**, ya que una vez que no queden objetos en la lista de visualización e intentemos seguir eliminando contenidos, Flash nos dará error. Para evitar este error, podemos utilizar el condicional **if(numChildren>0) removeChildAt(0)**, por medio del cual indicamos que se sigan eliminando contenidos siempre y cuando existan. En caso contrario, no se ejecutará la sentencia **removeChildAt()**.

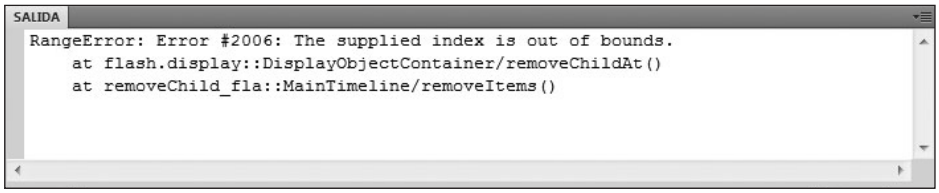


Figura 8. *Alerta de error de Flash al tratar de eliminar contenidos inexistentes dentro de un contenedor.*

Eventos: no más código en los botones

A diferencia de sus versiones antecesoras, ActionScript 3.0 presenta un único modelo de **gestión de eventos** que reemplaza a la modalidad empleada anteriormente. Este es otro de los grandes cambios que vamos a encontrar en este paso de AS2 a AS3: la sintaxis que utilizábamos anteriormente, **on (evento)**, ya no se emplea más en ActionScript 3.0. Esto implica que ya no se pueden utilizar códigos de eventos de AS dentro de clips de película o de botones. Si bien esto es difícil de asimilar para aquellos que aún programan de esta manera, es importante saber que esto acarrea dificultades: tener código en nuestros botones puede ser intuitivo y relativamente fácil de manejar en un proyecto muy pequeño que no requiera un gran nivel de interactividad. Pero no lo es si imaginamos un sitio con varias pantallas, todas ellas vinculadas de algún modo, y con una cantidad significati-

III PROPIEDADES GETCHILDAT(); Y GETCHILDBYNAME();

Si queremos seleccionar un objeto en particular dentro de la lista de elementos de visualización (**Display List**), podemos utilizar el método **getChildAt()** para seleccionarlo por su posición o emplear el método **getChildByName()** para seleccionarlo por su nombre.

va de funciones. Era realmente tedioso tener que navegar varios niveles hasta saber de dónde o hacia dónde se dirigían nuestros códigos. Afortunadamente, en AS3 esto ya no es así: podemos aplicar los eventos sobre nuestra línea de tiempo o dentro de las clases que utilizaremos luego.

Eventos y listeners

Dicho de manera sencilla, un **listener** es quien escucha para ejecutar ciertas acciones en respuesta a un evento. Para comprenderlo, daremos un ejemplo. En ActionScript 1 y 2 lo hacíamos como veremos a continuación. En caso de tener el código dentro de un botón:

```
on(release){
     trace("se acaba de liberar el botón");
}
```

En cambio, en caso de aplicarlo a una instancia de un MovieClip o un botón desde la línea de tiempo, debíamos hacer lo siguiente:

```
miClip.onRelease = function(){
     trace("se acaba de liberar el botón");
}
```

En AS3, el manejo de eventos (**event handling**) requiere de ciertas formalidades que si bien en un principio pueden resultar difíciles de incorporar, veremos que a largo plazo aportan un beneficio enorme, permitiéndonos:

- Tener un control mucho más exacto sobre nuestra película.
- Hacer un mejor uso de la memoria.
- No desperdiciar recursos.
- Optimizar nuestro código.

TIPOS DE EVENTOS

Existe la más diversa cantidad de eventos. Si bien algunos de ellos los utilizamos con mucha más frecuencia (eventos de mouse, por ejemplo), es importante saber que la mayoría de las clases de ActionScript tienen sus propios eventos. Existen eventos para manejar videos, MP3 y texto, entre otros que iremos viendo capítulo a capítulo.

Veamos la sintaxis para añadir un listener a un objeto:

```
miClip.addEventListener(MouseEvent.MOUSE_DOWN, ejecutarFuncion);
function ejecutarFuncion (event:MouseEvent):void{
     trace("se acaba de liberar el botón");
}
```

Analicemos línea por línea la estructura anterior. En primer lugar, podemos ver que definimos la instancia del MovieClip al cual le asignaremos un event listener. En nuestro caso, lo aplicaremos al MovieClip **miClip**. Luego agregamos el método **addEventListener()**, que contiene dos parámetros: el primero de ellos es el **evento** al cual queremos escuchar y el segundo es la **función** a la que debemos llamar cuando se detecte el evento. En nuestro caso, estamos esperando por el evento **MOUSE_DOWN** del mouse (**MouseEvent**) y la función que queremos ejecutar al producirse el evento es llamada **ejecutarFuncion**.

```
function ejecutarFuncion (event:MouseEvent):void{
     trace("se acaba de liberar el botón");
}
```

En primera instancia, puede resultar confuso este concepto, pero esta formalidad en el manejo de los eventos es una de las grandes cualidades de este nuevo lenguaje. A medida que avancemos en nuestro aprendizaje, veremos que tenemos un control mucho más exacto sobre los eventos y es muchísimo más sencilla la detección de errores. No debemos desesperar frente a esta nueva estructura; una vez que la incorporemos, será siempre la misma para cualquier tipo de evento. Hagamos un pequeño ejemplo para ponerlo en práctica.
En la **GUI** de **Flash** contamos con cinco clips: cuatro de ellos conformando un pequeño control con flechas para mover un reducido círculo en nuestro escenario, el quinto clip. Una vez instanciados (**upMc** para la flecha en sentido hacia arriba,

III RESPUESTA A EVENTOS

A diferencia de las funciones que empleamos generalmente, las que son en respuesta a un evento reciben información sobre el elemento que está ejecutando ese evento. El nombre lo podemos definir nosotros (**e**, **evt** o **event** son los más usados a modo de convención), y luego el tipo de elemento sobre el que estamos recibiendo la información, como un evento de mouse **MouseEvent**, por ejemplo.

downMc para la flecha en sentido hacia abajo y **mainClip** para el círculo), les agregamos a las flechas el método **addEventListener**, dentro del cual definimos el evento por el cual queremos escuchar y la función a ejecutar al detectarse ese evento. En ambos casos, esperamos por el evento **MOUSE_DOWN** y al detectarlo, dependiendo del clip, ejecutamos las funciones **setClipDown** o **setCLipUp**, las cuales se encargan de mover nuestro objeto sobre el **eje y** hacia arriba o hacia abajo.

```
upMc.addEventListener(MouseEvent.MOUSE_DOWN, setClipUp);
downMc.addEventListener(MouseEvent.MOUSE_DOWN, setCLipDown);

function setCLipDown(event:MouseEvent):void{
     mainClip.y +=5;
}

function setClipUp(event:MouseEvent):void{
     mainClip.y -=5;
}
```

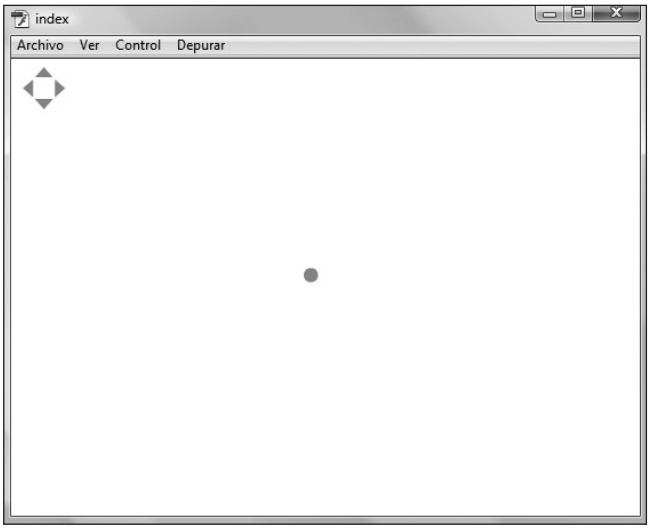


Figura 9. *Una vez exportada nuestra película, podremos ver los* ***Event Listeners*** *de nuestros MovieClips en funcionamiento.*

Como podemos notar, no están instanciadas las flechas en sentido hacia la izquierda y la derecha ni cuentan con sus respectivos listeners para mover el clip **mainClip** sobre el escenario. Por más de que sea una práctica menor, la intención de esto es que los lectores puedan agregar los eventos que faltan y sus respectivos códigos para ir interiorizándose con la sintaxis y el manejo de listeners.

KeyboardEvents (eventos de teclado)

Como mencionábamos anteriormente, una vez que dominamos el manejo de eventos en ActionScript 3.0, su funcionamiento es muy similar en todos los casos. Para comprobar esto, veremos un pequeño ejemplo de manejo de eventos de teclado, cuyo funcionamiento es prácticamente igual al de los eventos de mouse. Para realizar el siguiente ejemplo necesitamos tres MovieClips, los que instanciaremos como **aMc**, **sMc** y **tresMc**.

Figura 10. *El resultado final de nuestro desarrollo: cada clip responderá a un evento de teclado.*

Sobre la línea de tiempo, agregamos los siguientes listeners:

```
stage.addEventListener(KeyboardEvent.KEY_DOWN, keyIsDown);
stage.addEventListener(KeyboardEvent.KEY_UP, keyIsUp);
```

ASIGNAR EVENTOS

A diferencia de los eventos de mouse que podemos agregar a instancias de nuestros clips o botones, los eventos de teclado debemos incorporarlos sobre nuestro escenario (**stage**). De esta manera, estos eventos se van a ejecutar cada vez que presionemos un tecla (**KeyboardEvent.KEY_DOWN**) o cuando la liberemos (**KeyboardEvent.KEY_UP**).

En respuesta al Event Listener **KeyboardEvent.KEY_DOWN**, llamaremos a la función **keyIsDown**, como vemos en el siguiente código:

```
function keyIsDown(event:KeyboardEvent):void{switch(event.keyCode){
            case 65:
                  aMc.alpha = 0.2;
                  aMc.scaleX = aMc.scaleY = 1.5;
                  break;
            case 83:
                  sMc.alpha = 0.2;
                  sMc.scaleX = sMc.scaleY = 1.5;
                  break;
            case 51:
                  tresMc.alpha = 0.2;
                  tresMc.scaleX = tresMc.scaleY = 1.5;
                  break;
            case Keyboard.ENTER:
                  enterMc.alpha = 1;
                  break;
      }
}
```

Dentro de la función que ejecutamos al detectar el evento, podemos ver que contamos con la propiedad **keyCode**. **Keycode** es un identificador único que está asignado a cada tecla de nuestro teclado, de manera que filtrándolo podemos saber qué tecla es la que estamos presionando. En nuestro caso, utilizamos un **switch** para filtrar los valores y saber cuándo estamos presionando las teclas que son de nuestro interés: los valores **65**, **83** y **51**, respectivos valores de las teclas **A**, **S** y **3** de nuestro diseño. Al detectar cuando se presionan esas teclas (**KEY_DOWN**), modificamos tres de sus propiedades para resaltar el carácter que se está presionando, las cuales vuelven a su estado normal una vez que notamos el evento **KEY_UP**.

SOLUCIÓN EN CASO DE QUE EL TECLADO NO RESPONDA

Posiblemente, cuando ejecutemos nuestro código desde Adobe Flash CS4, la película no responda a los eventos de teclado. Si esto sucede, una vez exportada la película, debemos dirigirnos a **Control** dentro de la ventana de **Flash Player** y allí tildar la opción **Deshabilitar métodos abreviados de teclado**. Otra alternativa es exportar el archivo **SWF** y ejecutarlo fuera de la IDE de Flash.

```
function keyIsUp(event:KeyboardEvent):void{
     switch(event.keyCode){
            case 65:
                   aMc.alpha = 1;
                   aMc.scaleX = aMc.scaleY = 1;
                   break;
            case 83:
                   sMc.alpha = 1;
                   sMc.scaleX = sMc.scaleY = 1;
                   break;
            case 51:
                   tresMc.alpha = 1;
                   tresMc.scaleX = tresMc.scaleY = 1;
                   break;
     }
     if(enterMc.alpha == 1) enterMc.alpha = 0;
}
```

Por último, otro detalle importante de nuestro código es el uso de la clase **Keyboard**, que nos permite descubrir cuándo se presionan las teclas **ENTER**, **LEFT**, **RIGHT**, **UP**, **DOWN** y **SHIFT**, entre otras. Veamos a continuación:

```
case Keyboard.ENTER:
enterMc.alpha = 1;
break;
```

Eliminar listeners (removeEventListener)

Si bien puede no ser necesario eliminar un listener para un evento de botón e incluso quizás no debamos hacerlo ya que lo queremos funcionando constantemente, en muchas oportunidades vamos a requerir remover los Event Listeners que creamos. Por citar un ejemplo, supongamos que tenemos un **timer** que debe contar hasta un determinado número y luego ejecutar una acción.

Nuestro listener, una vez concluida la cuenta, ya no estará cumpliendo ninguna función, pero va a seguir contando repetitivamente y ocupando memoria, por lo que es una buena práctica eliminarlo. Para quitarlo, simplemente necesitamos del método **removeEventListener()**, al que debemos indicarle dos parámetros: el evento que deseamos dejar de escuchar y la función a la cual se hace referencia en ese evento.

Supongamos que contamos con un botón que queremos que solamente sea presionado una determinada cantidad de veces (cinco en nuestro caso). Para el ejemplo,

agreguemos sobre el escenario dos clips y asignémosle como nombre de instancia **oneMc** y **twoMc**. A su vez, coloquemos al lado de cada uno de ellos campos de texto dinámico, cuyos nombres de instancia sean **oneTxt** y **twoTxt**.

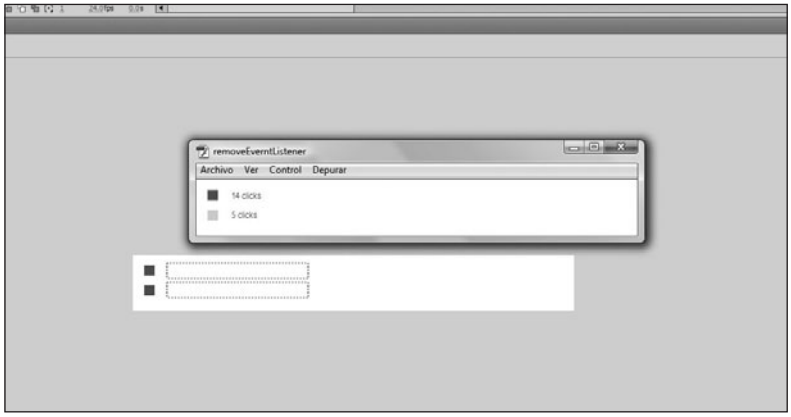


Figura 11. *Una vez concluida la cuenta, se elimina el listener que responde al evento, quedando funcional solamente nuestro primer botón.*

```
var oneUint:uint;
var twoUint:uint;

oneMc.addEventListener(MouseEvent.MOUSE_DOWN, oneMcMouseDown);
twoMc.addEventListener(MouseEvent.MOUSE_DOWN, twoMcMouseDown);

function oneMcMouseDown(event:MouseEvent):void{
     oneUint+=1;
     oneTxt.text = oneUint.toString() + " clicks";
}

function twoMcMouseDown(event:MouseEvent):void{
     trace("ingresando a la funcion");
     if(twoUint<5){
            twoUint+=1;
            twoTxt.text = twoUint.toString() + " clicks";
     }else{
            twoMc.alpha = 0.3;
            twoMc.removeEventListener(MouseEvent.MOUSE_DOWN, twoMcMouse
Down);
     }
}
```

Tenemos dos botones que, al ser presionados, van incrementando en uno el valor de dos variables del tipo **uint** (**oneUint**, **twoUint**). El primer botón queremos que cuente de forma indefinida y el segundo, que detenga la cuenta al llegar a 5. Para conseguir esto, simplemente eliminamos el listener del segundo botón de la manera que vemos a continuación:

```
if(twoUint<5){
     twoUint+=1;
     twoTxt.text = twoUint.toString() + " clicks";
}else{
     twoMc.alpha = 0.3;
     twoMc.removeEventListener(MouseEvent.MOUSE_DOWN, twoMcMouseDown);
}
```

Garbaje collector y el uso de la memoria

Si bien por el momento estamos utilizando solamente los dos parámetros necesarios para la creación de eventos, hay tres más que son de carácter opcional: **useCapture**, **priority** y **weakReference**. El uso del tercero de ellos nos puede ser de gran utilidad y representa una buena práctica. Empleando estos parámetros adicionales, la sintaxis del método **addEventListener** queda compuesta del siguiente modo:

```
target.addEventListener(TipoEvento.EVENTO, respuesta, userCapture:Boolean,
priority:int, weakReference:Boolean);
```

El parámetro **useCapture** nos da la posibilidad de controlar la fase del flujo del evento en la cual estará activa el detector. Difícilmente hagamos uso de este parámetro, por lo que dejaremos el valor **false** que viene por defecto.
El parámetro **priority** nos permite generar **prioridades** en el orden en el cual se organizan los detectores de nuestros eventos. Por medio de este parámetro podemos

LISTENERS Y USO CORRECTO DE LA MEMORIA

Debemos saber que cada **listener** ocupa una pequeña fracción de memoria. En un proyecto ActionScript de envergadura vamos a necesitar de varios de ellos y, en estos casos, no eliminarlos puede ocasionarnos problemas en la memoria que disponemos (**memory leak**). Es importante saber hacer un buen uso de los recursos de nuestra película.

asignar un valor entero (positivo o negativo), donde a mayor número mayor es su prioridad respecto al resto de los listeners con los cuales contamos. Por defecto, el valor de este parámetro es **0**, por lo que todos los detectores de eventos van a tener la misma prioridad y serán ejecutados en el orden en el que fueron añadidos al código. Al igual que el parámetro anterior, es difícil que necesitemos usarlo. En todos nuestros ejemplos dejaremos su valor por defecto.

El tercer y último parámetro es el que realmente nos interesa y haremos uso de él: **weakReference**. El manejo de memoria en ActionScript 3.0 está dado por lo que se denomina **garbaje collector** (recolector de basura). Cuando ya no se está haciendo referencia en el código a ningún objeto, el **garbaje collector** se encarga de eliminar esas referencias, liberando espacio de nuestra memoria.

Ahora bien, para que esto suceda, tenemos que definir como **true** el parámetro **weakReference**, de manera tal que cuando no hagamos más uso de determinado **eventListener**, éste no permanezca en memoria. De esta forma, se eliminan datos innecesarios que ya hemos dejado de usar. Este **garbaje collector** descarta de su **recolección** aquellos listeners que figuren con **weakReference** débil (**false**), por lo que si no modificamos manualmente su valor a **true**, no serán tenidos en cuenta a la hora de recolectar listeners fuera de uso para liberar memoria. El uso de este parámetro es verdaderamente sencillo, así que de aquí en adelante, crearemos nuestros listeners del siguiente modo:

```
target.addEventListener(EventType.EVENT_NAME, function, false, 0, true);
```

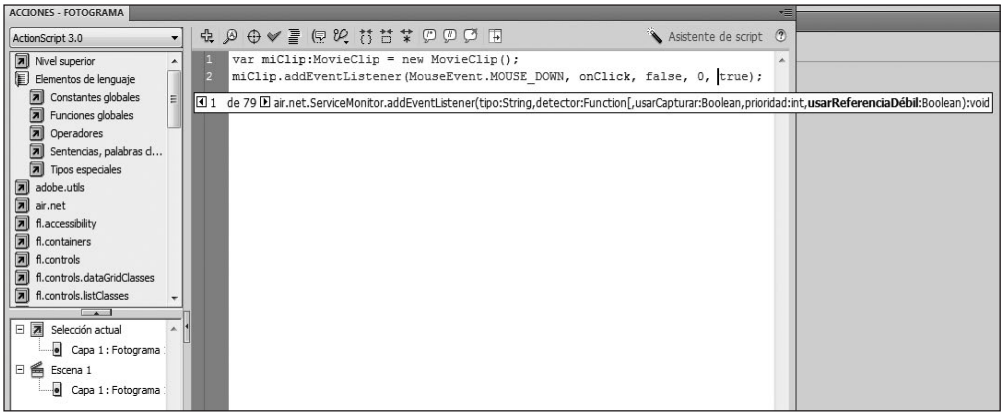


Figura 12. *Para acceder a weakReference, debemos indicar previamente valores para los otros parámetros: false para useCapture y 0 en priority.*

Si bien el parámetro que nos interesa modificar es el tercero de los opcionales, para poder hacer uso de éste debemos completar los valores para los otros dos, a los que les dejamos sus valores por defecto (**false** para **useCapture** y **0** para **priority**).

PROGRAMACIÓN ORIENTADA A OBJETOS

Esta introducción está orientada a explicar de forma comprensible qué es OOP (Object Oriented Programming) o POO, acrónimo en español de Programación Orientada a Objetos. En los comienzos de Flash, la escritura de código estaba limitada a una **secuencia lineal de instrucciones** donde se le indicaba al software, paso a paso, lo que debía a hacer. El principal problema con el que nos encontramos en este tipo de programación es su poca flexibilidad. Producto de la evolución del mismo lenguaje, se llegó a la **programación procedural**, que se basa en funciones o procedimientos, promoviendo un modo de desarrollo más **estructurado**. Esto facilitó que el código sea más reutilizable y fácil de editar.

Como resultado de esta continua evolución en pos de fomentar una más clara y mejor reutilización y legibilidad, se llegó a lo que se conoce actualmente como **Programación Orientada a Objetos**. La **POO** tiene similitudes con la **programación procedural**, diferenciándose en que agrupa los contenidos (variables, funciones, etcétera) en un elemento denominado **clase**.

La finalidad de la POO es **encapsular código** de manera que éste pueda ser independiente. La importancia de esto radica en que, en caso de tener que realizar cambios o correcciones en el desarrollo, modificando los contenidos de esa clase cumplimos nuestro cometido, sin tener que cambiar el resto de elementos. Esto hace mucho más fiable la escritura, el mantenimiento, la legibilidad y, por sobre todas las cosas, la **escalabilidad** de nuestros códigos, algo que era bastante más complejo de lograr con la programación procedural.

Si bien POO tiene grandes ventajas, entre ellas su enorme flexibilidad y reutilización, un mejor mantenimiento del código y una mayor prolijidad, no es necesariamente la solución a todos nuestros desarrollos. Es muy común que se tienda a creer que la Programación Orientada a Objetos es la clave para todos los desarrollos, y esto no siempre es así. POO seguramente sea la mejor solución para el desarrollo de grandes proyectos donde realmente se hace necesaria su implementación, pero puede ser innecesaria en proyectos de menor tamaño donde con programación procedural nos alcanza para resolver nuestros problemas sin necesidad de involucrarnos con clases y objetos.

GARBAJE COLLECTOR Y REMOVEEVENTLISTENER():

El uso del parámetro **weakReference** no reemplaza al método **removeEventListener**. Por el contrario, es un complemento de éste. De todos modos, en caso de que no utilicemos el método **removeEventListener**, sigue siendo una buena práctica emplear **weakReference**, evitando que queden referencias de listeners que ya no utilizamos.

Sintaxis de una clase

Para escribir una clase, debemos ir a **Archivo/Nuevo...** y, en la solapa **General**, elegimos la opción **Archivo ActionScript** y pulsamos **Aceptar**. Esto nos generará un archivo de **extensión .AS**, dentro del que debemos escribir el contenido de nuestra clase, respetando una lógica en su sintaxis:

```
package{

      import flash.display.Sprite;

      public class HolaMundo extends Sprite{

              public function HolaMundo():void{

                      trace(“Hola Mundo! :)” );

              }
      }
}
```

Vale mencionar que la estructura de una clase está compuesta, en primer lugar, por los **paquetes** (**package {**), que nos permiten organizar nuestro código en **grupos** de manera que luego podamos importarlos para utilizarlos.
A su vez, esto nos sirve para ser más prolijos y organizados con nuestros códigos, ya que por medio de los paquetes se hace verdaderamente sencillo **agrupar clases** que podamos asimilar por características en común. Al mismo tiempo, emplear paquetes nos evita conflictos con los nombres de las clases; no podemos tener dos clases con igual nombre dentro de un paquete pero sí podemos emplear clases con igual nombre en distintos paquetes. Imaginemos que tenemos que gestionar mediante clases el desarrollo de un formulario de registro de una universidad y debemos diferenciar el registro de docentes del de alumnos. Necesitaremos generar dos clases y ambas

SIN CLASES

Si no queremos adentrarnos en el manejo de las clases, la mayoría de los ejemplos de este libro cuentan con la escritura del código en sus dos versiones, tanto en clases como en líneas de tiempo. De todos modos, conocer y manejar clases es una enorme ventaja: pueden ahorrarnos mucho tiempo de desarrollo y brindarnos soluciones muy útiles en pocas líneas de código.

deberían llevar por nombre **Registro**, pero de ser así no podrían ir dentro del mismo paquete. Emplear paquetes nos solucionaría el inconveniente, permitiéndonos incluir la clase **Registro** destinada a alumnos dentro de un paquete denominado **alumnos** y la clase **Registro** destinada a docentes dentro del paquete denominado **docentes**. De este modo, tendríamos dos clases distintas con funcionalidades distintas pero con un mismo nombre, correctamente separadas. Manejarlas de esta manera nos permite una mejor organización del código y no nos limita con el nombre, ya que sabiendo a qué paquete pertenece, sabremos qué tipo de registro se encarga de gestionar cada clase.

En la segunda línea, **Import flash.display.Sprite**, por medio de la directiva **import**, cargamos todas las clases que vamos a utilizar a lo largo de nuestro desarrollo. Por medio de esa ruta, le indicamos al **compilador** de ActionScript dónde debe encontrar las clases que vamos a importar. Las clases preestablecidas de Flash están dentro del paquete **flash.***.

En algunas ocasiones necesitamos hacer uso de todas o de varias de las clases que contiene un paquete. En estos casos, en lugar de cargarlas una por una, podemos importar todas las clases del paquete por medio del comodín **.***. De todas maneras, si bien esto muchas veces resulta útil, puede convertirse en una mala práctica e incrementar las posibilidades de error. Por este motivo, lo más recomendable es mantener en nuestro código únicamente las clases con las que vamos a llevar a cabo nuestro desarrollo, y hacer un uso racional del comodín, implementándolo si en verdad necesitamos todas o varias de las clases de un paquete y no para ahorrar líneas de código al importar clases.

Nuestra próxima línea de código es **public class HolaMundo extends Sprite{**. En ella, lo primero que encontramos es el atributo **public**. Es importante tener en cuenta que las clases siempre deben ser públicas, ya que esto hace que estén disponibles para el resto de nuestro proyecto. Luego tenemos el nombre de nuestra clase, **HolaMundo**, y a continuación **extends Sprite**. Lo que hacemos al **extender una clase** a otra es habilitar, para nuestra clase, todos los eventos, métodos y propiedades de la clase que extendemos. En este caso, la clase **Sprite**. Es por este mismo motivo que la importamos luego de crear el paquete. Por último, como vemos en la próxima página, tenemos el constructor de nuestra clase:

CONVENCIONES RESPECTO A NOMBRES DE CLASES Y PAQUETES

A modo de convención, los nombres de los paquetes se escriben siempre con letra inicial minúscula, a diferencia de las clases que comienzan en mayúscula. Es muy importante tener presente esta práctica. No olvidemos que muchas veces otras personas deben continuar nuestro trabajo y respetar estas normas ayuda considerablemente a un mejor entendimiento y organización del código.

```
public function HolaMundo():void{
     trace("Hola Mundo! :)" );
}
```

El constructor es la primera fracción de código que se ejecutará al cargarse nuestra clase. Como vemos, debe llevar por nombre el mismo de la clase. Respecto a su contenido, no hemos hecho nada interesante por ahora; simplemente utilizamos la función **trace()** de Flash para hacer un **output** de una cadena de texto a fin de garantizarnos que la clase esté funcionando correctamente y que se haya ingresado al constructor de ella. De todas maneras, veremos a lo largo de los capítulos una clara tendencia a mantener la menor cantidad de código posible dentro del constructor, por lo que siempre llamaremos a otras funciones desde él para que realicen las tareas que necesitemos.

Vincular una clase a nuestro archivo .FLA (Document Class)

Vale mencionar que el concepto de **Document Class** es nuevo en ActionScript 3.0. Si bien contamos y podemos crear las más variadas clases, **Document Class** es la clase principal de nuestro desarrollo, que por defecto vinculamos a nuestro archivo .SWF. Cuando se carga nuestro .SWF, es llamado el **constructor** de esta clase. Una vez que creamos y guardamos nuestra clase, debemos vincularla a nuestro .FLA desde la IDE de Flash CS3 o CS4.

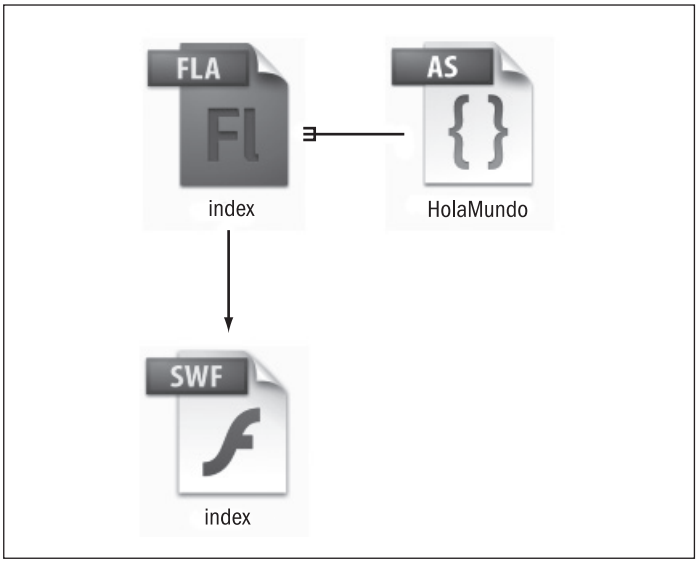


***Figura 13.** El archivo **.FLA** se vincula al archivo **.AS**. Cuando exportamos, se genera un archivo **.SWF**.*

Para definir esta clase como la principal de nuestro documento, lo hacemos desde el panel **PROPIEDADES**, indicando su nombre. Es importante recordar que lo que vincula al documento .FLA cuando definimos una clase principal es el nombre de la clase y no el archivo en sí. Es por este mismo motivo que, al indicar su nombre, no debemos agregar la extensión del archivo .AS al final de ésta. No siempre la clase se va a encontrar en el mismo directorio que nuestro archivo .FLA. En caso de que la clase se encuentre dentro de un subdirectorio, simplemente indicamos la ruta hasta llegar a ella. Por ejemplo, si se halla dentro de una carpeta denominada **com**, indicamos la ruta de la siguiente manera: **com.NombreClase**.

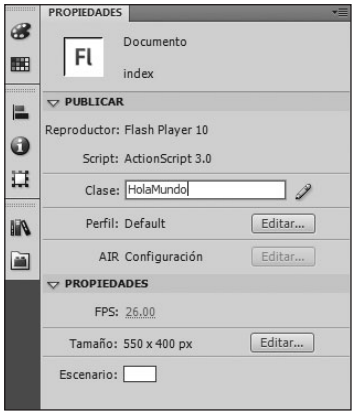


Figura 14. *Desde el panel **PROPIEDADES**, definimos el nombre de la clase. Es importante notar que lo que indicamos es el **nombre de la clase** y no **la ruta del archivo**. Por esto **no** indicamos la extensión .AS.*

A diferencia del resto de las clases que podamos llevar a cabo durante nuestro desarrollo, todas las que sean clase principal deben extender a la clase **Sprite** o a la clase **MovieClip**. Recordemos que llamamos clase principal a aquella que vinculamos a nuestro archivo .SWF ni bien se ejecuta la película.

... RESUMEN

La intención de este capítulo es sentar los conceptos básicos del lenguaje ActionScript 3.0 para poder afrontar los próximos desarrollos sin mayores problemas. Aquí conocimos el background teórico de ActionScript 3.0 y, en el siguiente capítulo, fijaremos todo lo que aprendimos pero de manera práctica, adentrándonos en el dibujo dinámico en Flash y en la creación de aplicaciones empleando la API de dibujo.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1) ¿Cuál es el cambio radical en ActionScript 3.0 respecto a sus versiones antecesoras?

2) ¿Cuáles son las principales características de este nuevo lenguaje?

3) ¿Cuáles son las diferencias en el manejo de variables entre ActionScript 3.0 y las versiones anteriores?

4) ¿Cuáles son las nuevas características de la sintaxis de las propiedades de los objetos y sus diferencias con ActionScript 2.0?

5) ¿Qué se entiende por DisplayList?

6) ¿Cuál es la diferencia entre Display Object y Display Object Container?

7) ¿Cuál es la sintaxis para agregar elementos al DisplayList? ¿Y para eliminarlos de ella?

8) ¿De qué manera se manejan los eventos en ActionScript 3.0?

9) ¿A qué se denomina Programación Orientada a Objetos?

10) ¿Qué es una clase?

EJERCICIOS PRÁCTICOS

1) Genere dos clips y agrégueles eventos para detectar cuando el mouse se encuentra sobre y cuando está fuera de estos.

2) Incorpore un símbolo de la biblioteca al escenario.

3) Agregue un evento para detectar cuándo se hace clic sobre los botones creados en el primer ejercicio práctico.

4) Al detectarse el evento creado en el ejercicio anterior:
-Modifique las propiedades **x**, **y**, **width**, **height** y **rotation** del símbolo que agregó en el segundo ejercicio.
-Haga que un botón incremente estos valores y otro los disminuya.

5) Coloque un nuevo botón que al hacer clic sobre él elimine del escenario el símbolo que agregó en el paso número 2.

6) Cree un nuevo documento de Flash (.FLA) y una nueva clase para él (.AS).

Dibujo en Flash

Una vez que conocimos los conceptos básicos de AS3, el mejor modo de ponerlos en práctica es por medio del desarrollo de ejemplos. En la primera parte veremos la API de dibujo vectorial en Flash para iniciarnos en el lenguaje, y crearemos una pizarra de dibujo al finalizar. En la segunda parte integraremos lo anterior con un lenguaje del lado del servidor (back-end) para poder almacenar nuestros dibujos como imágenes JPG en un servidor.

CREACIÓN DE UNA PIZARRA DE DIBUJO

Flash es una plataforma flexible que nos permite generar contenidos de diversas maneras. En lo que respecta al dibujo vectorial en Flash, generalmente se hace presente una gran duda entre qué es mejor: si dibujar por medio de **código** o por medio de las **herramientas** del programa. La respuesta a esta pregunta es que no existe una mejor o una peor manera de hacer las cosas. Existen distintas formas de hacerlas, siendo más o menos óptima una o la otra dependiendo de cada caso. Conocer cuándo es idónea una manera por sobre la otra va a facilitarnos, en gran medida, nuestros desarrollos. El beneficio de dibujar desde la **IDE** de Flash es que nuestros dibujos pueden tener cierta **complejidad**. Trabajamos de este modo cuando queremos hacer formas que serían prácticamente imposibles de representar por medio de programación. Por otra parte, la principal ventaja de dibujar por medio de **código** es la posibilidad de crear contenidos en **tiempo de ejecución** y la **optimización** de nuestras películas en cuanto a su peso. Respecto a la creación de contenidos en tiempo de ejecución, lo veremos en un ejemplo práctico por medio de la generación de una pizarra de dibujo, y a continuación, haremos una breve introducción a la optimización de recursos.

Optimización de recursos y pesos de archivos

Cuando hacemos dibujos desde la IDE de Flash, estos se almacenan en la **biblioteca** y ocupan espacio ahí dentro. En cambio, cuando dibujamos por medio de código, las formas se generan en tiempo de ejecución, por lo que no están ocupando espacio dentro de nuestra película. Para verlo mediante un ejemplo, crearemos una misma figura desde la IDE de Flash y por medio de código. A un archivo lo llamaremos **dibujoCodigo.fla** y al otro **dibujoHerramienta.fla**.

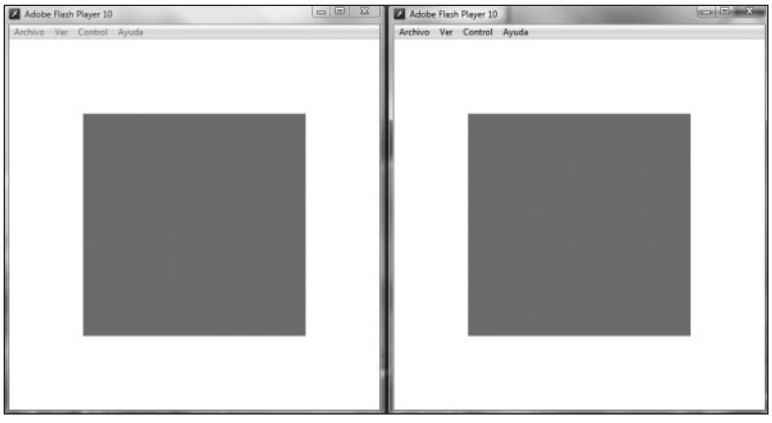


Figura 1. *Visualmente, ambos archivos son iguales tanto en el tamaño del canvas (500 px x 500 px) como en el del rectángulo que se encuentra dentro de estos (300 px x 300 px), y son del mismo color.*

Ahora bien, si observamos la información referente al tamaño de cada archivo, veremos que son diferentes. Contamos con dos archivos cuyas resoluciones finales son idénticas, pero uno pesa menos que el otro.

Figura 2. *El peso de los archivos difiere:* ***dibujoCodigo.swf*** *pesa* ***970 bytes*** *y* ***dibujoHerramienta.swf*** *pesa* ***1012 bytes****.*

La diferencia entre estos archivos es que en **dibujoCodigo.swf**, el rectángulo que se encuentra en la escena lo creamos por medio de código, de la siguiente manera:

```
var dibujo:Sprite = new Sprite();
dibujo.graphics.beginFill(0xff0066, 1);
dibujo.graphics.drawRect(0, 0, 300, 300);
dibujo.graphics.endFill();
dibujo.x = dibujo.y = 100;
addChild(dibujo);
```

OPTIMIZACIÓN DE PELÍCULAS

El correcto manejo de los pesos y la adecuada optimización de nuestros desarrollos es vital para obtener el mejor rendimiento de nuestras películas. No debemos perder de vista que no todos los usuarios visitan nuestros sitios o utilizan nuestras aplicaciones con un ancho de banda considerable como para desligarnos de estas cuestiones.

En cambio, en el archivo **dibujoHerramienta.swf**, el rectángulo que se encuentra en escena lo hemos dibujado utilizando las herramientas del programa, y luego lo añadimos al escenario de la siguiente manera:

```
var cuadrado:Cuadrado = new Cuadrado();
cuadrado.x = cuadrado.y = 100;
addChild(cuadrado);
```

A medida que tenemos una mayor cantidad de contenidos, más elevado será el peso de nuestras películas SWF. Por esta razón, es importante conocer y manejar estos conceptos para lograr una mejor **optimización**.

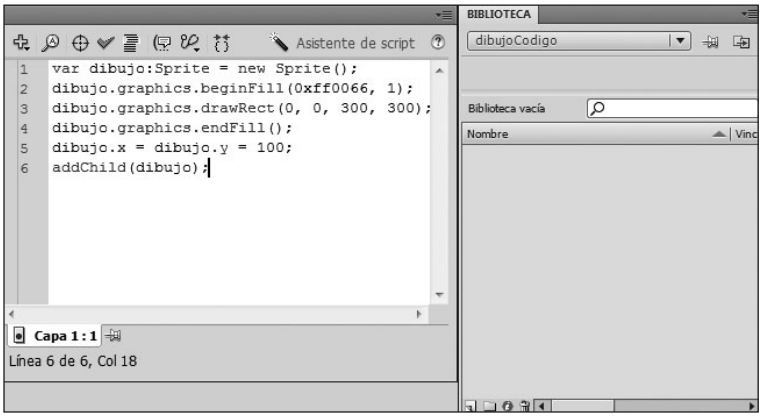


Figura 3. *En dibujoCodigo, el contenido lo dibujamos dinámicamente, por lo que su BIBLIOTECA no contiene ningún elemento.*

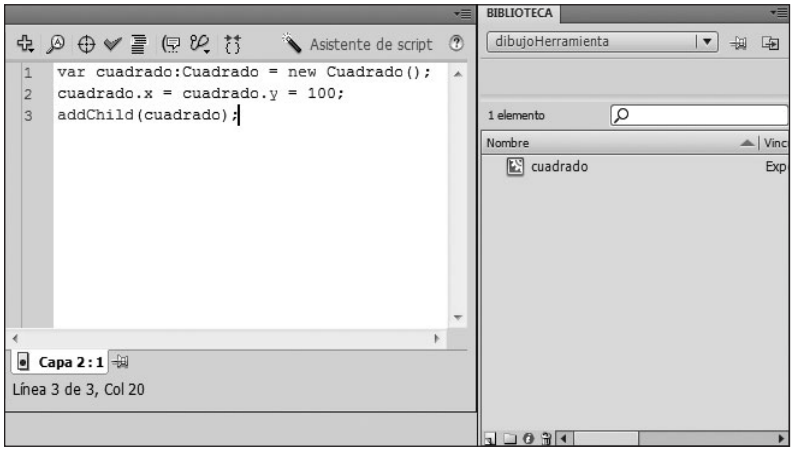


Figura 4. *En dibujoHerramienta, vinculamos al escenario un elemento de la BIBLIOTECA dibujado desde la IDE de Flash.*

Dibujo en Flash en tiempo de ejecución

Desde la IDE de Flash, contamos con la barra **HERRAMIENTAS** (si no la visualizamos, debemos presionar **Ctrl + F2** o ir a **Ventana/Herramientas**). Desde esta barra podemos generar diversas formas vectoriales, como círculos, cuadrados, líneas y trazados. Todas estas formas que son producto del dibujo vectorial desde la IDE de Flash, también las podemos representar mediante código haciendo uso de la clase **Graphics**. Esta clase nos permite generar cualquier tipo de dibujo vectorial, ya sean líneas, gradientes, rectángulos o círculos. Cada vez que utilicemos la clase **Graphics**, tenemos que hacerlo dentro de un **contenedor**, que debemos definir y sobre el que aplicamos las transformaciones:

```
var dibujo:Sprite = new Sprite();
dibujo.graphics.beginFill.[métodos];
```

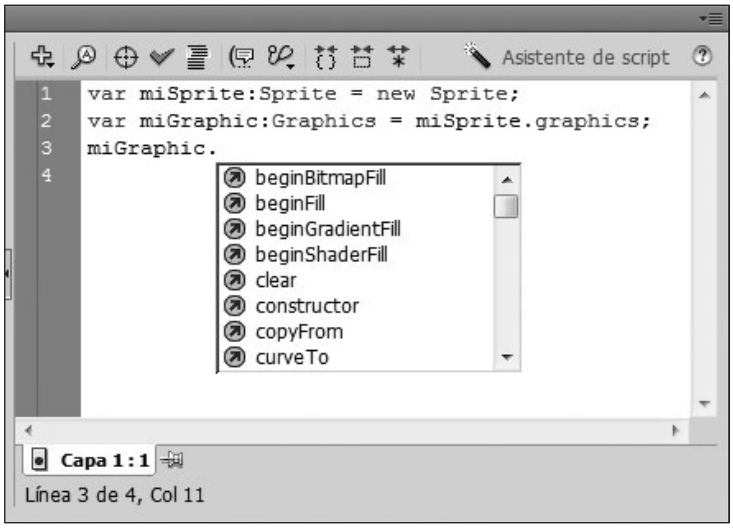


***Figura 5.** Una vez creada la instancia de la clase Graphics del objeto de visualización, cada vez que la escribamos veremos todos sus métodos y las alternativas de dibujo vectorial con las que contamos.*

III API DE DIBUJO DE FLASH

Cuando nombramos la **API de dibujo de Flash**, estamos refiriéndonos, justamente, a la clase **Graphics** y a todos sus métodos, por medio de los cuales podemos generar nuestros dibujos vectoriales. Como encontraremos este término en reiteradas ocasiones a lo largo de este capítulo, es importante saber bien a qué se hace referencia al nombrarla.

Dibujo de rectángulos: graphic.drawRect

La clase **Graphics** contiene métodos que hacen realmente sencilla la creación de formas. Por ejemplo, para generar un rectángulo, debemos emplear el método **drawRect** de la manera que vemos a continuación:

```
var miSprite:Sprite = new Sprite;
var miGraphic:Graphics = miSprite.graphics;
miGraphic.beginFill(0xff0066, 1);
miGraphic.drawRect(0, 0, 50, 50);
addChild(miSprite);
```

Analicemos los nuevos conceptos de este código. En primer lugar, antes de crear un rectángulo, debemos definir su color de relleno. Esto lo hacemos por medio del método **beginFill()**, asignando dos parámetros: el **color** y la **transparencia**. Recordemos que en ActionScript 3.0, los valores de alpha van de 0 a 1 y no de 0 a 100 como en sus versiones antecesoras. La línea es **miGraphic.beginFill(0xff0066, 1);**.
Luego, al generar el rectángulo por medio de **drawRect**, tenemos que indicar cuatro parámetros: la posición **x**, la posición **y**, el **ancho** (**width**) y el **largo** (**height**). Por ejemplo, **miGraphic.drawRect(0, 0, 50, 50);**. Al ejecutar nuestra película, se crea un rectángulo en tiempo de ejecución y se lo añade al escenario.
Una de las mayores confusiones al dibujar formas vectoriales es no distinguir entre el objeto de visualización sobre el que estamos trabajando y su instancia de la clase **Graphics**. Esto se hace notorio a la hora de posicionar los elementos sobre el escenario. Por ejemplo, si cambiamos la línea anterior por **miGraphic.drawRect(40, 40, 50, 50);**, veremos que el rectángulo que generamos aparece sobre las coordenadas x:40 e y:40.
Ahora bien, que el dibujo comience en la posición x:40 e y:40 no implica que nuestro objeto de visualización (**miSprite**) se encuentre en esa posición. Éste sigue ubicado sobre las coordenadas x:0 e y:0, que es la posición por defecto de un elemento al ser agregado al escenario y no asignársele una ubicación para dicho contenedor. Observemos que en el código no indicamos coordenadas para **miSprite**. Hagamos otra prueba para entenderlo de forma más clara. Agreguemos coordenadas a **miSprite**:

```
var miSprite:Sprite = new Sprite;
miSprite.x = miSprite.y = 40;
```

Ahora, al ejecutar la película, nuestro rectángulo comienza en la posición x:80 e y:80. Es importante no confundir estos conceptos: una cosa es el objeto de visualización sobre el cual trabajamos y otra es la forma que creamos dentro de él utilizando la clase **Graphics**. Veamos el resultado en la siguiente imagen:

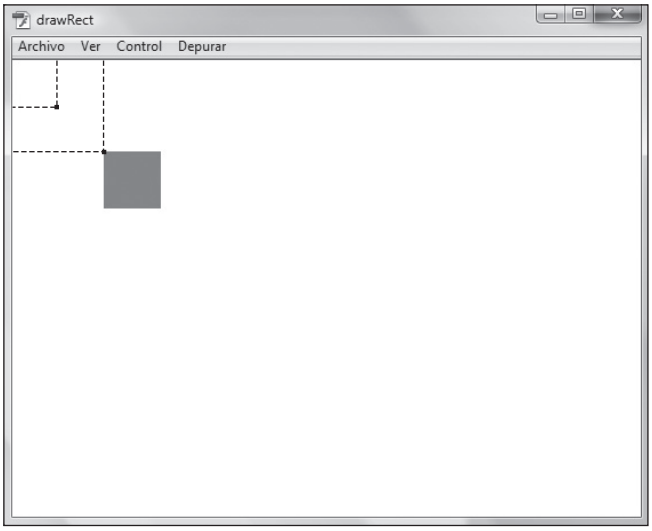


Figura 6. *Si bien visualmente el rectángulo se encuentra en x:80, y:80, esta ubicación es la suma de las posiciones de nuestro sprite* ***miSprite*** *(x:40, y:40) y del dibujo que creamos dentro de él:* ***miGraphic*** *(x:40, y:40).*

Dibujo de círculos: graphic.drawCircle

La creación de círculos es muy similar a la de rectángulos en lo que a código respecta. La diferencia es que, lógicamente, al generar un rectángulo, debemos indicar un ancho y un largo y al crear círculos, tenemos que establecer el radio de la circunferencia: **miGraphic.drawCircle(posicionX, posicionY, radio);**. En este caso, nuestro código es muy similar al anterior:

```
var miSprite:Sprite = new Sprite;
miSprite.x = miSprite.y = 60;
var miGraphic:Graphics = miSprite.graphics;
miGraphic.beginFill(0xff0066, 1);
miGraphic.drawCircle(0, 0, 40);
addChild(miSprite);
```

III ATAJOS EN FLASH: NUEVO SHORTCUT PARA LA LUPA

Históricamente, la letra **Z** representó el **shortcut** de la **lupa** en Flash, pero este atajo en Flash CS4 fue asignado a la herramienta de **huesos**, siendo la letra **M** el nuevo atajo para la lupa. Si usamos Flash desde hace años y no nos acostumbramos a este cambio, la buena noticia es que estos atajos de teclado se pueden modificar o eliminar desde el menú **Edición/Métodos abreviados de teclado…**.

Otros métodos

Si bien los círculos y los rectángulos son las formas que con mayor frecuencia vamos a dibujar mediante código, también contamos con otros métodos que, aunque su uso es menos frecuente, nos pueden ser de utilidad para algunos casos específicos.

Dibujar elipses:

```
miGraphic.drawEllipse(0, 10, 80, 20);
```

Donde los parámetros a indicar son x, y, anchura y altura.

Dibujar rectángulos con puntas redondeadas:

```
miGraphic.drawRoundRect(0, 0, 50, 50,20);
```

Utilizamos los cuatro parámetros que empleamos para la creación de rectángulos y contamos con un quinto parámetro, en el que indicamos el redondeo de las puntas.

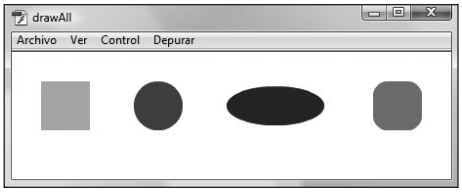


Figura 7. *En el archivo drawAll.fla tenemos cuatro sprites con sus respectivos gráficos de cada uno de los métodos que vimos: drawRect, drawCircle, drawEllipse y drawRoundRect.*

Si bien los métodos que utilizamos aquí son los que se emplean con más frecuencia, también nos resultará útil saber que contamos con procedimientos como **drawPath()**, **drawGraphicsData()** y **drawTriangles()**. Estos tienen funciones más específicas y forman parte, al igual que los métodos hasta aquí vistos, de la API de dibujo de Flash.

¿QUÉ SON LAS APIS?

Encontraremos el acrónimo **API** (**Application Programming Interface**) en reiteradas oportunidades. Las APIs son un conjunto de funciones de las que hacemos uso sin necesidad de programar sus contenidos por nosotros mismos. Por ejemplo, un rectángulo lo podríamos dibujar por medio de cuatro líneas, pero en lugar de eso utilizamos el método **drawRect**.

CREACIÓN DE UNA PIZARRA

Este ejemplo nos será de gran ayuda para asimilar y aplicar los conceptos referentes al manejo de eventos en ActionScript 3.0 que aprendimos en el **capítulo 2**.

Dibujo de líneas

Del mismo modo que podemos crear formas utilizando la API de dibujo, también podemos generar **líneas** o **bordes** para nuestras formas. La sintaxis y el modo de crearlas son verdaderamente sencillos. Antes de comenzar a generar una línea, debemos definir una serie de características: su **grosor**, el **color** y algunos parámetros adicionales de ser necesario (la **transparencia** del trazo, por ejemplo). Para asignar estos parámetros, lo hacemos por medio del método **lineStyle**.

```
var miLinea:Sprite = new Sprite();
miLinea.graphics.lineStyle(1, 0x00ff00);
```

Figura 8. *Cuando escribimos el método y abrimos paréntesis, Flash nos muestra todos los* ***parámetros****,* ***obligatorios*** *y* ***opcionales****, que podemos asignar.*

Una vez que contamos con nuestro estilo, podemos comenzar con el dibujo de nuestras líneas. La mejor manera de pensar el dibujo en Flash es de la misma forma en la que lo hacemos en la vida real sobre un papel. Puede sonar muy obvio pero lo primero que debemos hacer para comenzar a dibujar es apoyar el lápiz sobre el papel. El equivalente a esta acción en ActionScript es el método **moveTo**:

```
var miLinea:Sprite = new Sprite();
miLinea.graphics.lineStyle(5, 0x00ff00,1);
miLinea.graphics.moveTo(60, 60);
```

Una vez que **apoyamos el lápiz**, debemos hacer que se desplace hasta la posición que deseemos. En ActionScript, esto lo efectuamos por medio del método **lineTo**. A continuación, podemos ver un ejemplo para comprenderlo mejor.

```
var miLinea:Sprite = new Sprite();
miLinea.graphics.lineStyle(5, 0x00ff00,1);
miLinea.graphics.moveTo(60, 60);
miLinea.graphics.lineTo(190, 60);
addChild(miLinea);
```

Esta pequeña teoría es todo lo que se requiere para comenzar a crear nuestra pizarra. Antes de empezar su desarrollo, analicemos los comportamientos y acciones necesarias:

1. Si bien podríamos considerar diversas alternativas para dibujar por medio del puntero del mouse, en nuestro ejemplo los trazos se generarán mientras mantengamos el botón del ratón presionado y lo movamos sobre el escenario. Con este criterio, hagamos una analogía con el mundo real: cuando dibujamos con un lápiz sobre un papel, para comenzar a dibujar, es necesario apoyar nuestro lápiz sobre el papel. Traducida al mundo virtual, esta acción equivaldría a presionar el botón izquierdo del mouse sobre el escenario. Esto nos da la pauta de que debemos detectar por medio de un evento de mouse cuándo el botón correspondiente es presionado (**MOUSE_DOWN**). Mientras esta acción se mantenga, sabemos que cada movimiento debe generar un trazo.
2. Los trazos deben seguir el puntero del mouse, por lo que su destino final (**lineTo**) siempre será a la posición en la cual se encuentre el mouse (**mouseX** y **mouseY**).
3. Debemos detectar cuándo se libera el botón del mouse ya que en este caso tenemos que dejar de dibujar. Esto también lo haremos por medio de un evento de mouse; en este caso, **MOUSE_UP**.

Ahora sí, veamos paso a paso cómo generar nuestra pizarra. En primer lugar, creamos el sprite que contendrá nuestro dibujo y definimos el estilo de nuestra línea:

```
var miLinea:Sprite = new Sprite();
miLinea.graphics.lineStyle(10, 0x00ff00, 1);
addChild(miLinea);
```

Luego, asignamos los eventos de mouse para detectar cuando el botón se presionó y cuando se liberó. En base a esto, sabemos cuándo dibujar y cuándo dejar de hacerlo.

```
stage.addEventListener(MouseEvent.MOUSE_DOWN, posicionarCursor, false, 0,
true);
stage.addEventListener(MouseEvent.MOUSE_UP, liberarCursor, false, 0, true);
```

Al detectarse el clic, definimos la posición desde la cual vamos a comenzar a dibujar (**mouseX** y **mouseY**, las coordenadas del mouse) y agregamos el evento **ENTER_FRAME**, que llamará constantemente a la función **dibujar**.

```
function posicionarCursor(event:MouseEvent):void{
     miLinea.graphics.moveTo(mouseX, mouseY);
     addEventListener(Event.ENTER_FRAME, dibujar);
     Mouse.hide();
}
```

La función **dibujar()** genera permanentemente una línea desde el último punto donde se encuentre nuestro trazado hasta el puntero de nuestro mouse. De esta manera, se genera el **trazo continuo**. La línea de código comentada se encarga de modificar el color del trazo y, en caso de **descomentarla**, cada vez que se ejecute el evento nuestra línea cambiará de color, quedando un efecto bastante curioso. En caso contrario, la línea continuará en el color que definimos al crear el estilo de línea.

```
function dibujar(event:Event):void{
     miLinea.graphics.lineTo(mouseX, mouseY);
     //miLinea.graphics.lineStyle(10, Math.random()*0xffffff, 1);
}
```

Al liberar el cursor, eliminamos el **listener** que se encarga de llamar a la función **dibujar**, de modo que no se generarán más trazos.

```
function liberarCursor(event:MouseEvent):void{
     removeEventListener(Event.ENTER_FRAME, dibujar);
     Mouse.show();
}
```

III OCULTAR Y MOSTRAR EL CURSOR DEL MOUSE

A la hora de dibujar los trazos, se suele ocultar el cursor y reemplazarlo por otro puntero (un aerosol, por ejemplo). Con **Mouse.hide()** lo ocultamos y con **Mouse.show()** volvemos a hacerlo visible. Para que un objeto siga la posición del mouse mientras se realizan los trazos, hacemos que sus posiciones **x** e **y** sean iguales a las posiciones **x** e **y** del mouse (**mouseX** y **mouseY**).

Figura 9. *La variación de tonos es producto de haber* ***descomentado*** *la línea de código encargada de generar un nuevo estilo de línea (**lineStyle()**) cada vez que se llama a la función **dibujar()**.*

Borrar el dibujo

Del mismo modo que podemos crear trazos, también es posible borrarlos. Esto lo hacemos por medio del método **clear()**. Agreguemos un clip a nuestra pizarra que, al ser presionado, se encargue de eliminar nuestro dibujo. Una vez que incorporamos nuestro clip al escenario e indicamos su nombre de instancia (**borrar**), dentro del código colocamos su respectivo listener:

```
borrar.addEventListener(MouseEvent.MOUSE_DOWN, borrarDibujo, false, 0,
true);
```

Al presionarlo, llamamos a la función **borrarDibujo**, que es la que se encarga de eliminar las líneas del último dibujo realizado, y define nuevamente el estilo de línea para que podamos seguir utilizándolo.

```
function borrarDibujo(event:MouseEvent):void{
     miLinea.graphics.clear();
     miLinea.graphics.lineStyle(10, 0x00ff00, 1);
}
```

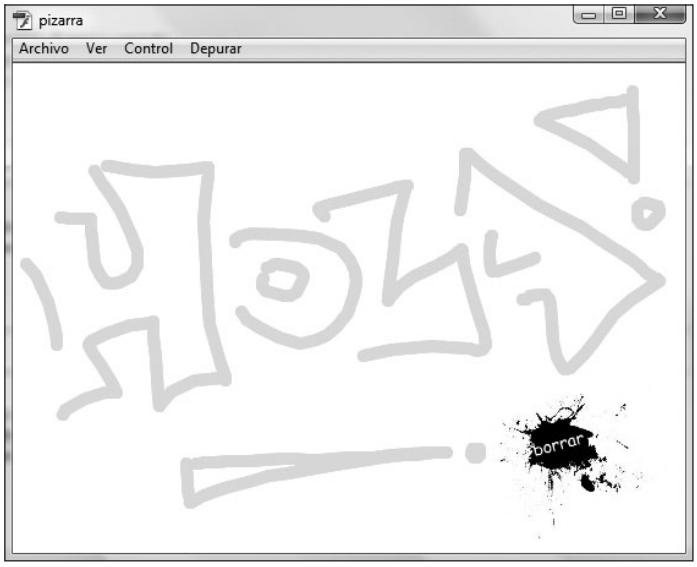


Figura 10. *Nuestra pizarra en funcionamiento con nuestros trazos y la implementación del botón para borrar los dibujos.*

Ya hemos visto, en esta primera parte del capítulo, los conceptos básicos de dibujo en Flash, y los empleamos para crear una pizarra simple que cuenta con las funcionalidades básicas para dibujar y borrar nuestros trazos. En la segunda parte de este apartado, veremos de qué manera podemos integrar Flash con otras tecnologías para poder convertir nuestros dibujos en imágenes.

ALMACENAR NUESTROS DIBUJOS: INTEGRACIÓN CON PHP

Antes de comenzar este proyecto, como éste es nuestro primer capítulo práctico, es importante que recordemos que los temas se distribuyen en **dos partes**. En la primera parte vimos los conceptos básicos referidos al tema principal del capítulo, orientados a aquellos usuarios que estén conociendo el lenguaje y se estén adentrando en el mundo de Flash. En la segunda parte abarcaremos aspectos más profundos y complejos del tema central del capítulo, generalmente extendiendo lo hecho en la primera parte. Esta sección está principalmente enfocada en la comunidad de desarrolladores o usuarios avanzados de Flash, por lo que no analizaremos las cuestiones más básicas del lenguaje ni de los procedimientos que llevaremos a cabo. Por ejemplo, a continuación integraremos una aplicación realizada en Flash con PHP, pero no nos detendremos en su instalación ni en sus características.

Entonces, veremos de qué manera integrar una aplicación desarrollada en Flash (cliente) con uno de los tantos lenguajes del lado del servidor que hay (PHP). Es importante recordar que el objeto de estudio de este libro es Flash y no PHP ni ninguna otra tecnología; por lo que simplemente veremos de qué manera abordar los conceptos necesarios para aplicar de forma correcta la sintaxis de ActionScript 3.0 a la hora de querer enviar y recibir variables con un servidor.

Flash + PHP: integración

Una de las grandes ventajas de Flash, quizás una de las más importantes a la hora de hablar de desarrollo, es su potencial para poder integrarse y comunicarse con **lenguajes** del lado del **servidor**. PHP es tan sólo uno de los tantos **lenguajes back-end** con los que Flash puede comunicarse, pero es el que emplearemos en este apartado para poder guardar nuestras imágenes en el servidor o bien para facilitar la descarga de los dibujos de la pizarra a través de una imagen.

Flash

El archivo con el que trabajaremos a lo largo de este apartado lo adaptamos a una clase a la que llamaremos **Main.as**. Es posible suponer que los desarrolladores que alcancen esta parte del capítulo estarán más familiarizados con el manejo de clases que con el código dentro de frames de la línea de tiempo. De todos modos, vale aclarar que este paso de una forma a otra es por una cuestión de practicidad. Si preferimos seguir programando desde la línea de tiempo, también podemos hacerlo.

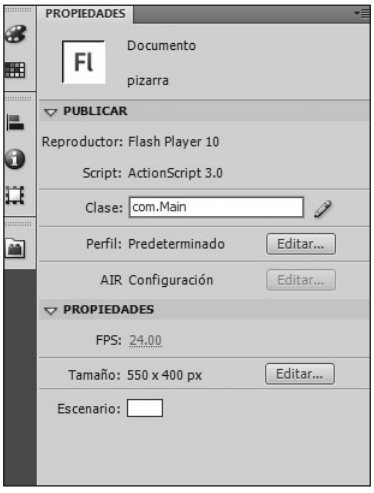


Figura 11. *Recordemos que cuando trabajamos con una* ***Main Class*** *o clase principal, debemos indicarle al archivo .FLA la ruta de ésta. Esto lo hacemos desde el panel* ***PROPIEDADES****, sección* ***PUBLICAR****.*

Veamos el código de **Main.as**. Si bien el código con el que generamos nuestros dibujos es exactamente el mismo, encontraremos líneas nuevas: algunas son necesarias para el correcto funcionamiento de las clases y otras son parte de la nueva interfaz con la cual guardaremos los dibujos. En primer lugar, definimos el paquete dentro del cual se encuentra nuestra clase **Main.as** con **package com{**. Luego, a diferencia de cuando programamos desde la línea de tiempo, debemos importar las clases que vamos a utilizar en nuestro código:

```
import flash.display.Sprite;
import flash.display.BitmapData;
import flash.events.MouseEvent;
import flash.events.Event;
import flash.ui.Mouse;
import flash.utils.ByteArray;
import flash.net.URLRequest;
import flash.net.URLRequestHeader;
import flash.net.URLRequestMethod;
import flash.net.URLLoader;
import flash.net.URLLoaderDataFormat;

import com.adobe.images.*;
import com.adobe.file.*;
```

Algunas de estas clases ya las hemos visto y otras las iremos conociendo a lo largo de la explicación de nuestro código. Lo más importante aquí es notar que no todas las clases que importamos son de Flash: las últimas dos, que se encuentran dentro del paquete Adobe, corresponden a un conjunto de clases perteneciente a un proyecto denominado **corelib**. Corelib es un proyecto desarrollado en ActionScript 3.0 conformado por una librería de clases y utilidades que, básicamente, nos permiten realizar tareas complejas con una gran sencillez (manejo de **encoders** para imágenes, **serialización** de **JSON**, **MD5** y **SHA 1 hashing**, entre otras cosas). La página

III PHP EN DETALLE

Si aún no dominamos el lenguaje PHP en profundidad, pero estamos interesados en lograrlo, podemos consultar en Internet o leer otras publicaciones de esta misma editorial. Entre las últimas, encontraremos **Curso de programación PHP**, **PHP MASTER** y **PHP 5**, obras escritas por Francisco José Minera.

oficial del proyecto es **http://code.google.com/p/as3corelib/** y desde ella podemos realizar la descarga del archivo que contiene este paquete de clases. De todos los paquetes que contiene este proyecto, solamente necesitamos dos de ellos: el paquete **images** y el paquete **file**, con todos sus contenidos.

***Figura 12.** Sitio web del proyecto **as3corelib**. Allí encontraremos mucha información de interés.*

A continuación creamos la clase **Main**, que contendrá todo el código que necesitamos para nuestra pizarra. Veámoslo a continuación:

```
public class Main extends Sprite {
```

Dentro de la clase **Main** definimos las variables que vamos a utilizar en nuestro código. Como veremos, ya conocemos y usamos anteriormente estas variables, excepto la última, que explicaremos en breve.

III ESCRITURA DE CLASES Y NOMBRES PARA PAQUETES

Como mencionamos en el segundo capítulo, es importante recordar que, por convención, los nombres de las clases comienzan en mayúscula, a diferencia de los paquetes (carpetas) que las contengan, que empiezan en minúscula. Por ejemplo, **com/Main.as**, donde **com** es el directorio que contiene a la clase nombrada **Main**.

```
private var contenedor:Sprite;
private var miLinea:Sprite;
private var guardar:Guardar;
private var borrar:Borrar;
private var miBitmapData:BitmapData;
```

Luego creamos el constructor de la clase y, dentro de él, llamamos a la función **init()**, que es la que se encarga de generar nuestra interfaz.

```
public function Main():void {
     init();
}

private function init():void {

     miLinea = new Sprite();
     miLinea.graphics.lineStyle(5, 0x00ff00, 1);
     addChild(miLinea);

     stage.addEventListener(MouseEvent.MOUSE_DOWN, posicionarCursor, false,
0, true);
     stage.addEventListener(MouseEvent.MOUSE_UP, liberarCursor, false, 0,
true);

     borrar = new Borrar();
     borrar.rotation = 20;
     borrar.x = 420;
     borrar.y = 270;
     borrar.addEventListener(MouseEvent.MOUSE_DOWN, borrarDibujo, false, 0,
true);
     addChild(borrar);

     guardar = new Guardar();
     guardar.rotation = -20;
     guardar.y = 320;
     guardar.addEventListener(MouseEvent.MOUSE_DOWN, enviarDibujo, false, 0,
true);
     addChild(guardar);
}
```

El botón **borrar** continúa realizando su misma función, mientras que agregamos un nuevo botón a nuestra interfaz, que lleva por nombre de instancia **guardar**. Seguramente, ya sabemos cuál va a ser su función. Ésta es la parte más interesante de nuestro código y la de mayor importancia, por lo que explicaremos cada una de las líneas de código que componen la función **enviarDibujo()**, que se ejecuta al presionar el botón que lleva por nombre de instancia **guardar**:

```
private function enviarDibujo(event:MouseEvent):void{
     miBitmapData = new BitmapData(550, 400, false);
     miBitmapData.draw(miLinea);
     var byteArray:ByteArray = new JPGEncoder( 90 ).encode( miBitmapData );
     var urlRequest:URLRequest = new URLRequest();
     urlRequest.url = "http://localhost/pizarra/php/image.php";
     urlRequest.contentType = 'multipart/form-data; boundary=' + Upload
PostHelper.getBoundary();
     urlRequest.method = URLRequestMethod.POST;
     urlRequest.data = UploadPostHelper.getPostData('file.jpg', byteArray );
     urlRequest.requestHeaders.push( new URLRequestHeader( 'Cache-Control',
'no-cache' ) );
     var urlLoader:URLLoader = new URLLoader();
     urlLoader.dataFormat = URLLoaderDataFormat.BINARY;
     urlLoader.load(urlRequest);
}
```

A continuación conoceremos y nos familiarizaremos con algunos conceptos nuevos de ActionScript 3.0 (las clases **URLRequest**, **URLLoader** y **BitmapData**), y también haremos uso de las clases del proyecto **corelib** que importamos, desde donde enviaremos las imágenes que tomemos de nuestra pizarra a PHP.
Como podemos ver, encontramos un nuevo concepto: la clase **BitmapData**. El mejor modo de comprenderla es pensándola como un contenedor de un conjunto de datos de píxeles. Para usar esta clase, en primer lugar, necesitamos de-

III CLASES BITMAP() Y BITMAPDATA()

No debemos confundir las clases **Bitmap()** y **BitmapData()** ya que, si bien se complementan, son dos clases distintas. Por medio de **BitmapData** almacenamos información de datos de píxeles y a través de la clase **Bitmap** convertimos esos datos en un mapa de bits.

finir sus dimensiones. En este caso, los valores que asignamos son 550 y 400, que son las mismas dimensiones de nuestra película:

```
miBitmapData = new BitmapData(550, 400, false);
```

Luego, por medio del método **draw**, dibujamos el objeto de visualización (el sprite **miLinea**) en la imagen de mapa de bits que acabamos de generar por medio del constructor **New BitmapData()**.

```
miBitmapData.draw(miLinea);
```

A continuación, hacemos uso de una de las clases del paquete **corelib**:

```
var byteArray:ByteArray = new JPGEncoder(90).encode(miBitmapData);
```

Es importante que entendamos esta línea. Primero, creamos una instancia para una nueva clase que se introdujo en ActionScript 3.0, la clase **ByteArray**, que proporciona propiedades y métodos para la lectura y escritura de datos binarios. No nos adentraremos en este tema ya que si bien es de gran interés, comprender su funcionamiento y sus métodos requeriría de varias páginas de escritura. Lo que sí es necesario saber es que dentro de ese ByteArray almacenaremos toda la información **codificada** de nuestra imagen. Para hacer esto, utilizamos la clase **JPGEncoder** y, posteriormente, el método **encode** de esa clase:

```
new JPGEncoder(90).encode(miBitmapData);
```

Al definir la clase, el parámetro que asignamos equivale a la **calidad** final que tendrá nuestra imagen, que establecimos en **90** para nuestro ejemplo (sus valo-

EJEMPLO DE USO DE LA API DE DIBUJO

En el sitio de la compañía de publicidad de Leo Burnett (**www.leoburnett.com**) encontraremos un gran ejemplo de lo que se puede lograr con un buen uso de la API de dibujo de Flash. La lógica no difiere mucho de lo que vimos en este capítulo. Es interesante ver cómo, con un excelente concepto de diseño, es posible potenciar los recursos que nos brinda la programación en ActionScript.

res van del 1 al 100). A continuación, el parámetro para **.encode()** es el objeto **BitmapData** que queremos codificar para que, una vez almacenado en nuestro ByteArray, podamos enviarlo al servidor. Para finalizar, veamos los últimos conceptos que vamos a incorporar en este capítulo:

```
var urlRequest:URLRequest = new URLRequest();
```

Allí, la clase **URLRequest** se encarga de capturar toda la información en una única solicitud HTTP. Dentro de ella indicamos, por medio del método **url**, la URL a la cual se envía la información, como vemos en la siguiente línea:

```
urlRequest.url = "http://localhost/pizarra/php/image.php";
```

A través del método **contentType** indicamos, en la cabecera, de qué tipo es el archivo que estamos enviando. En este caso, un archivo **binario**:

```
urlRequest.contentType = 'multipart/form-data; boundary=' + UploadPostHel
per.getBoundary();
```

Por medio de **method**, especificamos el método con el que enviaremos los contenidos al servidor: **POST** en este caso.

```
urlRequest.method = URLRequestMethod.POST;
```

Valiéndonos del método **data**, enviamos la foto codificada.

```
urlRequest.data = UploadPostHelper.getPostData('name.jpg', byteArray );
```

III COMPRESIÓN DE IMÁGENES EN CORELIB

Si bien en este ejemplo estamos haciendo uso de la clase **JPGEncoder()**, dentro del **paquete com.adobe.images** encontraremos otra clase denominada **PNGEncoder()** que como su nombre lo indica, a diferencia de **JPGEncoder()**, genera imágenes **PNG** en lugar de imágenes JPG. Entre otras cosas, la ventaja de este formato es que soporta **transparencia**.

Así como empleamos **URLRequest** para almacenar toda la información para la solicitud HTTP, utilizamos la clase **URLLoader** para descargar datos desde una URL, ya sea como texto, datos binarios o variables con codificación del tipo URL.

```
var urlLoader:URLLoader = new URLLoader();
urlLoader.dataFormat = URLLoaderDataFormat.BINARY;
urlLoader.load(urlRequest);
```

PHP

Si no manejamos PHP, es importante saber que, para que el servicio funcione, es necesario instalarlo en la computadora o bien probar los ejemplos online, en algún **hosting** que soporte PHP. A continuación brindamos algunos links respecto a algunas aplicaciones para instalar el servidor Apache y PHP bajo Windows o MAC. A estas aplicaciones se las conoce como **WAMP** (**Windows**, **Apache**, **MySQL**, **PHP**) o **MAMP** (**MAC**, **Apache**, **MySQL**, **PHP**) y su instalación nos permitirá usar nuestro propio equipo como servidor para poder probar desde él las aplicaciones que creamos en PHP. Si bien a continuación hay una tabla que contiene algunos WAMPs y MAMPs, no explicaremos en este libro cómo se procede a su instalación, ya que escapa a nuestro tema central que es Flash. Por esto mismo, es importante recordar que la aplicación que generamos **no funcionará** en caso de no tener un servidor Apache corriendo.

NOMBRE	TIPO	URL
Apache2Triad	WAMP	**http://apache2triad.net**
AppServ	WAMP	**www.appservnetwork.com**
EasyPHP	WAMP	**www.easyphp.org/index.php**
XAMPP	WAMP	**www.apachefriends.org/en/xampp.html**
MAMP	MAMP	**www.mamp.info/en/index.html**
XAMPP	MAMP	**www.apachefriends.org/en/index.html**

***Tabla 1.** Listado de los **WAMPs** y **MAMPs** más utilizados.*

DISTINTOS MODOS DE ALMACENAMIENTO DE IMÁGENES

Si bien enviamos las imágenes desde Flash y las almacenamos por medio de PHP en formato JPG, también podríamos guardar, en una base de datos, las coordenadas de los trazos del dibujo para luego tomarlos desde Flash y recrearlos. Aunque la primera alternativa es más práctica y quizás más útil, la segunda, pese a su complejidad, puede generar un gran impacto visual.

Como sabemos, no es PHP la prioridad de este libro, por lo que el código que utilizamos es lo más sintético y legible posible:

```
<?PHP

$path = "../dibujos/".uniqid("flash_extremo_", false).".jpg";

if ( move_uploaded_file( $_FILES[ 'Filedata' ][ 'tmp_name' ], $path) ){
//aquí el código que necesitemos en caso de
//requerirse alguna función al moverse la imagen..
}
?>
```

Como vemos, creamos una variable llamada **path** para nuestra imagen, a la cual le indicamos la ruta en la que debe almacenarse el dibujo. Todas nuestras ilustraciones se guardarán en la carpeta **dibujos**, que se encuentra un directorio debajo de la carpeta **PHP**, por lo que debemos indicarlo en la ruta:

```
$path = "../dibujos/"
```

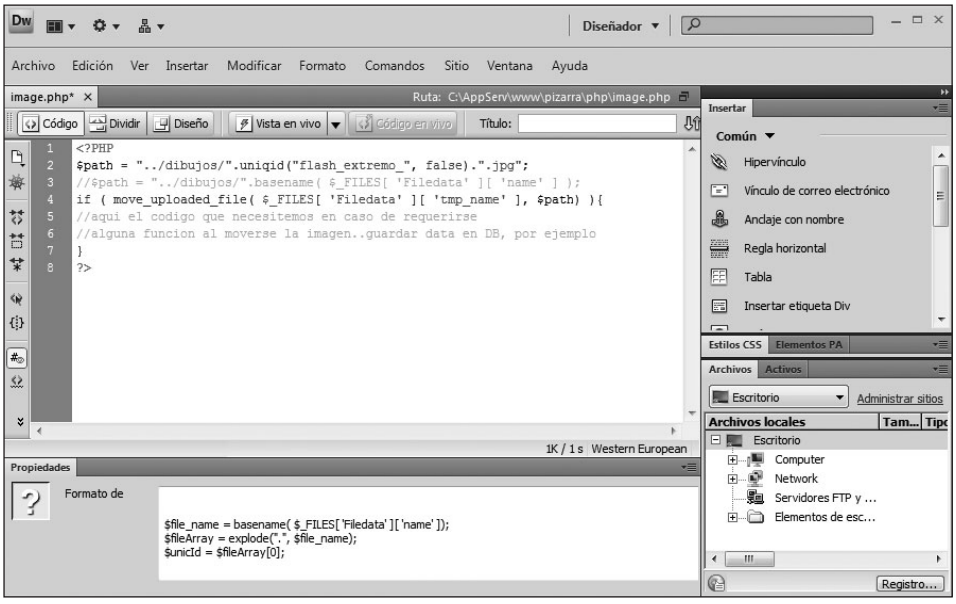


Figura 13. *En Dreamweaver, vemos nuestro código PHP para este ejemplo. Si bien simplificamos su sintaxis a lo mínimo e indispensable, las posibilidades que nos brinda este lenguaje son muchísimas.*

Luego le concatenamos un nombre único para cada imagen. Esto lo hacemos por medio de la función **uniqid()** de PHP, la cual generará un **valor único** e irrepetible para cada dibujo. Cuando trabajamos con este tipo de aplicaciones, es imprescindible evitar que los nombres de las imágenes se repitan, lo que traería problemas en el servidor, renombrando los archivos ya existentes o generando errores. A su vez, esta función también nos permite agregarle un prefijo al valor que generará. En nuestro caso, ese valor será **flash_extremo**.

En último término, debemos mover la imagen al directorio **../dibujos** por medio de **move_uploaded_file**. Dentro del condicional podemos emplear el código que sea necesario una vez que se subió y almacenó la imagen correctamente. Cuando se ejecute el código PHP, nuestros dibujos se irán guardando en la carpeta **dibujos**.

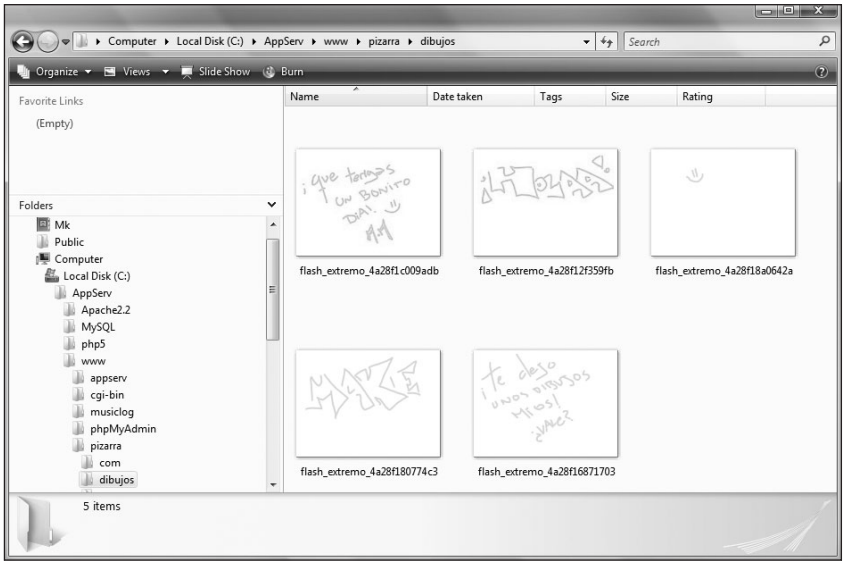


Figura 14. *La carpeta dibujos con sus respectivas imágenes .JPG, que creamos por medio de PHP y almacenamos dentro del directorio. Todos los dibujos que se creen se moverán a esta carpeta.*

... RESUMEN

El dibujo por código en Flash puede ser de gran utilidad, no sólo para optimizar recursos en nuestras películas sino para poder generar aplicaciones en tiempo de ejecución, siendo un ejemplo de esto la pizarra dinámica que creamos en este capítulo. A su vez, el lenguaje nos facilita alternativas para potenciar nuestros desarrollos, principalmente si logramos implementar Adobe Flash CS4 junto con otros lenguajes.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1) ¿Qué es la API de dibujo de Flash?

2) ¿Cuál es la diferencia entre el dibujo vectorial desde la IDE de Flash y el dibujo por medio de código?

3) ¿Cuáles son las ventajas del dibujo por medio de código? ¿Y desde la IDE de Flash?

4) ¿Por medio de qué método de la clase **Graphics** se generan rectángulos?

5) ¿Qué métodos hay para crear círculos, elipses o rectángulos con puntas redondeadas?

6) ¿Con qué método definimos el estilo para las líneas?

7) ¿Cuáles son los parámetros que podemos asignar a un estilo de línea?

8) ¿Con qué método nos posicionamos en un punto para comenzar a dibujar y con cuál nos movemos hasta un determinado punto?

9) ¿Con qué método borramos un dibujo?

10) ¿Qué es corelib? ¿Qué clases de este proyecto utilizamos?

11) ¿Para qué sirven las clases **URLRequest**, **URLLoader** y **BitmapData**?

EJERCICIOS PRÁCTICOS

1) Genere su propia pizarra de dibujo y asigne los estilos de línea que crea convenientes.

2) Agregue un botón que le permita modificar el grosor de los trazos de sus dibujos.

3) Incorpore un botón que brinde la posibilidad de modificar el color del trazo.

4) Agregue un último botón que le permita cambiar la transparencia de los trazos.

5) En caso de haber generado su código desde la línea de tiempo, adapte ese código a una clase e inclúyala en su archivo .FLA.

Manejo de texto en Flash

En la primera parte de este capítulo veremos los conceptos necesarios para manejar correctamente el texto, optimizando los recursos y las alternativas que Flash propone, tanto desde la IDE del programa como por medio de ActionScript. En la segunda parte integraremos lo aprendido al comienzo para generar un formulario de contacto, implementando Flash con PHP, para enviar las variables vía e-mail.

CONCEPTOS BÁSICOS SOBRE MANEJO DE CAMPOS DE TEXTO

En esta primera parte del capítulo abarcaremos los conceptos básicos para hacer un mejor uso del texto en Flash, tanto desde la IDE como por medio de programación en ActionScript 3.0. El mismo modo que vimos en capítulos anteriores es válido para este caso en particular: podemos usar las herramientas de texto de la IDE de Flash para la creación de campos de texto y también es posible hacerlo por medio de código, siendo ventajoso el uso de una u otra modalidad dependiendo de cada caso en particular. Para empezar, determinaremos cuándo nos resulta conveniente llevar a cabo nuestro desarrollo de una manera o de otra y veremos que, si bien para todo lo que realicemos desde la IDE tenemos un equivalente en código, hay algunas operaciones que solamente las podemos realizar por medio de ActionScript.

Campos de texto desde la IDE de Flash

Como hemos visto, podemos generar campos de texto tanto desde la IDE de Flash como por medio de código. Desde la IDE del programa contamos con tres tipos de campos de texto: **Texto estático**, **Texto dinámico** e **Introducción de texto**. A continuación veremos en detalle cada uno de ellos.

Texto estático

Sobre un campo de texto estático, una vez creada y exportada la película, no podemos realizar ningún tipo de modificación. Antes de eso, sí es posible definir algunas propiedades básicas, como el espacio entre letras, el tamaño de la fuente, el color o el tipo de suavizado (desde el panel **CARÁCTER**) o el formato, espaciado, márgenes y orientación (desde el panel **PÁRRAFO**). El uso de este tipo de campo es recomendable cuando tenemos **bloques de texto** sobre los que sabemos que, una vez exportada nuestra película, no vamos a realizar ninguna modificación; tal como su nombre lo indica, es un texto estático.

III NO EMBEBER FUENTES EN TEXTO ESTÁTICO

Una de las ventajas al trabajar con texto estático es que no necesitamos embeber la fuente que estamos empleando en nuestro desarrollo, ya que Flash las **renderizará como vectores**. En cambio, al trabajar con campos de texto dinámicos sí debemos embeber las fuentes. Próximamente veremos de qué manera hacerlo.

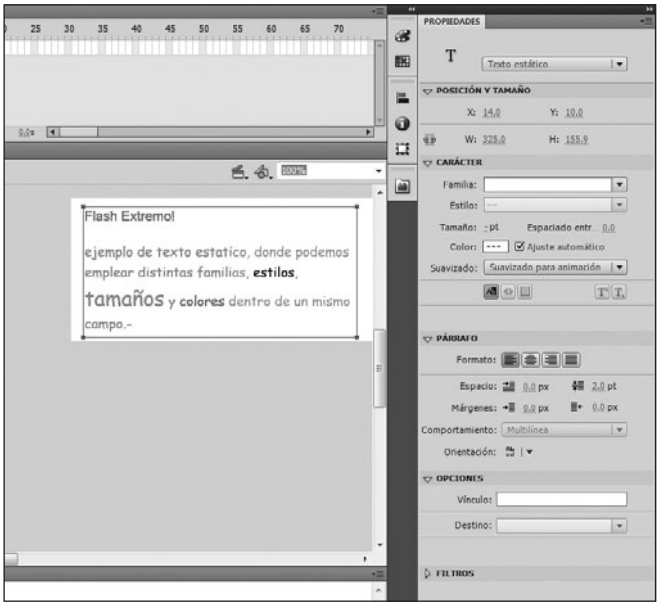


Figura 1. *Al seleccionar un texto estático, en el panel* ***PROPIEDADES*** *vemos las opciones para este tipo de texto:* ***POSICIÓN Y TAMAÑO, CARÁCTER, PÁRRAFO, OPCIONES*** *y* ***FILTROS.***

Texto dinámico

La característica de este tipo de texto es que, una vez que le asignamos un **nombre de instancia**, podemos aplicar las más diversas modificaciones a los contenidos del campo y al formato del texto cuando ya se exportó la película. Desde la IDE, Flash nos facilita algunas opciones más respecto a los textos estáticos. Por ejemplo, en el panel **PROPIEDADES** contamos con los iconos **Generar texto como HTML** y **Mostrar borde alrededor del texto** debajo de las opciones de **Suavizado**. Al exportar una película con un campo de texto dinámico, en caso de no utilizar **fuentes** del sistema, debemos **embeberlas**. Podemos usar la opción **Relleno automático** o embeber los juegos de caracteres que deseemos.

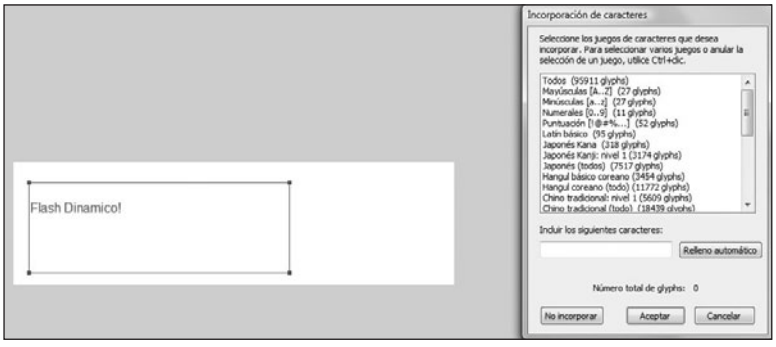


Figura 2. *Al crear campos de textos dinámicos, debemos embeber sus fuentes en caso de no ser fuentes de dispositivo.*

Introducción de texto

Como su nombre lo indica, empleamos este tipo de campo de texto cuando esperamos que el **usuario escriba** sobre él. Si bien algunas de las acciones que podemos realizar sobre este tipo de campo son muy similares a las que es posible llevar a cabo sobre un campo dinámico, sus funciones son distintas. Podríamos considerarlo como un campo de texto dinámico que puede ser editado por el usuario.

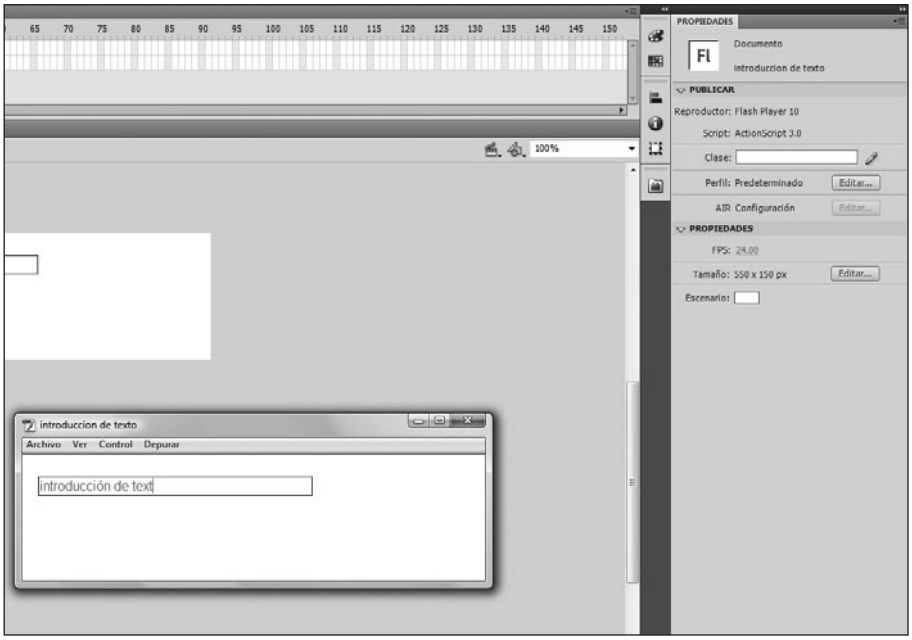


***Figura 3.** Un campo de introducción de texto posee, básicamente, las mismas propiedades y métodos que un campo de texto dinámico. La diferencia es que el usuario puede editar su contenido.*

Como ya adelantamos, en algunos casos es útil el uso de campos de texto desde la IDE de Flash, pero, en otros, estos recursos nos limitan completamente. Supongamos que debemos adaptar el ancho o el largo de una columna de texto a una medida variable, que necesitamos que el largo de un campo de texto sea **autoajustable** a

III CAMPOS DE TEXTO Y CANTIDAD DE CARACTERES POR FILA

Salvo en casos experimentales, recordemos que la finalidad primaria es que el texto pueda ser leído sin dificultades por parte de los usuarios. Existen análisis e informes en los que se establece la cantidad de palabras por fila de texto para que se lean los contenidos sin cansar al usuario. Tengamos en cuenta que no es lo mismo leer contenidos desde una pantalla que sobre un papel.

la cantidad de caracteres de la cadena o que los contenidos que se van a generar son completamente aleatorios. La IDE de Flash no nos proporciona alternativas para este tipo de problemas, por lo cual debemos emplear la programación para encontrar las soluciones. Por esto, es importante que conozcamos y dominemos las dos maneras de trabajar con texto y que sepamos distinguir cuándo nos conviene hacerlo de una forma y cuándo de otra. A continuación veremos cómo crear campos de texto mediante código y cómo aplicarles formatos, y nos iremos adentrando en cuestiones que hacen a un mejor control de los contenidos, independientemente de cómo hayan sido creados.

CREACIÓN DE UN FORMULARIO DE CONTACTO

Cuando hacemos mención al uso de texto en Flash, no necesariamente es un formulario la única aplicación en la que vemos reflejada el uso del texto, pero sí es la más viable y la que nos permite conocer más a fondo todo el entorno de programación que hace al manejo de campos de textos.

Creación de campos de texto en ActionScript 3.0

A continuación conoceremos la sintaxis necesaria para crear campos de texto de forma dinámica, es decir, mediante ActionScript. A su vez, iremos viendo sus ventajas sobre los campos de texto que generamos desde la IDE de Flash. Para crear campos de texto por medio de ActionScript, lo hacemos empleando la clase **TextField**:

```
var miTextField:TextField = new TextField();
```

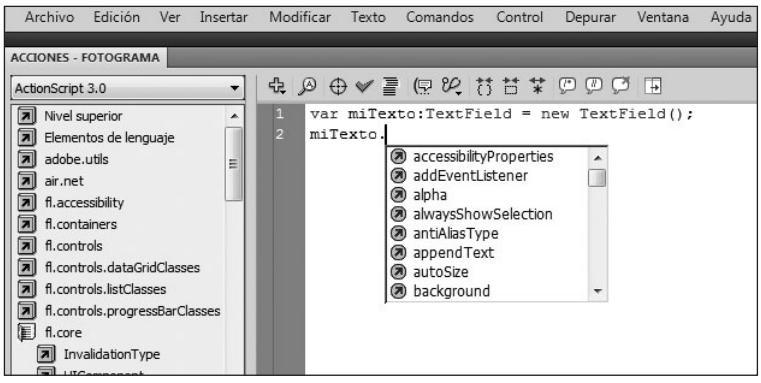


Figura 4. *Al escribir el nombre de la* ***instancia*** *en nuestro código, podremos ver todas las* ***propiedades*** *y métodos del objeto.*

Analicemos, en primer lugar, los campos que no son del tipo **Introducción de texto** (**nombreEst**, **emailEst**, **asuntoEst**, **formEst**). Este es todo el código que necesitamos para crear un campo de texto dinámicamente:

```
nombreEst = new TextField();
nombreEst.embedFonts = true;
nombreEst.defaultTextFormat = formato;
nombreEst.text = “nombre:”
nombreEst.selectable = false;
nombreEst.autoSize = TextFieldAutoSize.RIGHT;
nombreEst.textColor = 0x666666;
nombreEst.x = 10;
nombreEst.y = 30;
nombreEst.width = 100;
nombreEst.height = 14;
mainContainer.addChild(nombreEst);
```

Una vez definido el **constructor**, podemos emplear todos los métodos y propiedades que hereda para crear y manipular los textos en tiempo de ejecución. Las principales propiedades de las que hacemos uso son las siguientes:

- Propiedad **text**: establece el texto que irá en nuestro campo.

```
nombreEst.text = “nuestro primer texto por codigo...”;
```

- Propiedad **textColor**: define el color del texto.

```
nombreEst.textColor = 0x666666;
```

- Para modificar las dimensiones del campo de texto, podemos hacerlo de dos maneras. Por medio del uso de las propiedades **width** y **height**:

```
nombreEst.width = 100;
nombreEst.height = 14;
```

O bien, utilizar la propiedad **autoSize** y emplear la clase **TextFieldAutoSize** para su alineación. De esta manera, logramos que las dimensiones del campo se ajusten a la

cantidad de texto que contamos. No tendríamos manera de lograr esto utilizando un campo de texto desde la IDE de Flash.

• Propiedad **selectable**: por defecto, su valor es **true**, pero podemos modificarlo.

```
miTexto.selectable = false;
```

Los campos de texto también cuentan con las propiedades básicas para manipularlos en escena:

```
nombreEst.x = 10;
nombreEst.y = 30;
```

Además existen las propiedades **alpha**, **visible**, **rotation**, **scaleX** y **scaleY**, que estamos acostumbrados a usar por ser propiedades de **MovieClips** y **Sprites**.
Sigamos avanzando y veamos otro campo creado mediante código. En este caso, será un campo del tipo **Introducción de texto** (**nombreInput**). Como podremos ver, presenta algunas variaciones respecto al campo dinámico que hemos creado anteriormente (**nombreEst**):

```
nombreInput = new TextField();
nombreInput.defaultTextFormat = formato;
nombreInput.type = TextFieldType.INPUT;
nombreInput.textColor = 0x666666;
nombreInput.x = 110;
nombreInput.y = 30;
nombreInput.width = 170;
nombreInput.height = 13;
nombreInput.border = true;
nombreInput.borderColor = 0xD0D0D0;
nombreInput.background = true;
nombreInput.backgroundColor = 0xE1E1E1;
mainContainer.addChild(nombreInput);
```

Hay una propiedad en particular que los diferencia:

```
nombreInput.type = TextFieldType.INPUT;
```

Para definir un campo de texto como válido para la introducción de texto, debemos hacerlo por medio de su propiedad **type**, y para asignarle un valor a esta propiedad debemos usar la clase **TextFieldType**. Sus valores pueden ser **INPUT** o **DYNAMIC**. Al definirlo en **INPUT**, nuestro campo pasa a ser un campo de tipo **Introducción de texto**. Veamos que a este código también le aplicamos varias propiedades más, las cuales no son solamente atribuibles a los campos del tipo **Introducción de texto**, sino a cualquier campo de texto creado mediante código:

• Propiedades **background**, **backgroundColor**, **border** y **borderColor**:

```
nombreInput.border = true;
nombreInput.borderColor = 0xDODODO;
nombreInput.background = true;
nombreInput.backgroundColor = 0xE1E1E1;
```

Al establecer el valor de **background** en **true**, hacemos que el fondo del campo sea visible y luego, por medio de **backgroundColor**, indicamos su color. El concepto es el mismo pero aplicado a los bordes del campo para **border** y para **borderColor**.
Podemos definir un campo como multilínea por medio de la propiedad **multiline**, y con **wordwarp** hacemos que el texto se ajuste a su caja. A continuación, veamos cómo utilizar estas propiedades:

```
formInput.multiline = true;
formInput.wordWrap = true;
```

Existen otras propiedades y métodos que iremos viendo a lo largo del capítulo, pero la intención de este breve repaso es conocer los que se emplean con más frecuencia a la hora de trabajar el texto por medio de código. Muchos de ellos los iremos descubriendo con su uso en este mismo capítulo, y también es conveniente que investiguemos sobre los otros, ya que pueden resultarnos de utilidad.

III SELECCIONAR UNA PARTE DEL TEXTO: MÉTODO SETSELECTION()

El método **setSelection()** se utiliza para definir la cantidad de caracteres que deseamos que aparezcan seleccionados al hacer foco en el campo de texto. Usamos **índices** para indicar desde dónde y hasta dónde seleccionar: **campoDeTexto.setSelection(6, 13);**. De este modo, se seleccionará desde el carácter 6 hasta el 13 de un campo de texto.

Embeber fuentes en ActionScript

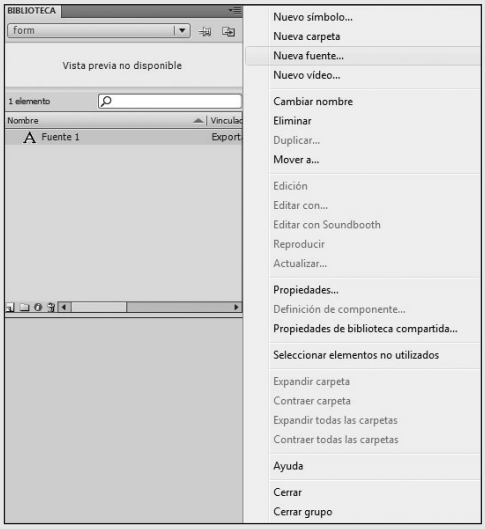
Si bien conocimos de qué manera embeber fuentes en campos dinámicos al trabajar desde la IDE de Flash, no hemos visto la manera de hacerlo al programar un formulario por medio de ActionScript. Para nuestro ejemplo, embebimos la fuente **FFF Harmony**; ésta es una tipografía gratuita que descargamos desde el sitio **www.fontsforflash.com**, donde encontraremos gran variedad de fuentes para Flash, la mayoría de ellas son pagas pero otras son gratuitas. Estas fuentes, denominadas **pixel fonts** (o **fuentes pixelares**), fueron especialmente diseñadas para que cada borde de cada uno de los caracteres de la familia encaje con exactitud en los bordes de los pixeles de la pantalla. Si bien esto les confiere una gran legibilidad y una bonita apariencia en Flash, para hacer un buen uso de ellas es necesario respetar algunas particularidades; generalmente se utilizan a 8 puntos o a múltiplos de 8 y deben posicionarse sobre coordenadas x e y sin decimales.

Es importante saber que la decisión de implementar fuentes pixelares hace a la legibilidad y a la apariencia de nuestro formulario, pero podemos utilizar la tipografía que deseemos. En este caso, usamos esta fuente para enriquecer nuestro desarrollo y ver de qué modo embeber fuentes mediante código.

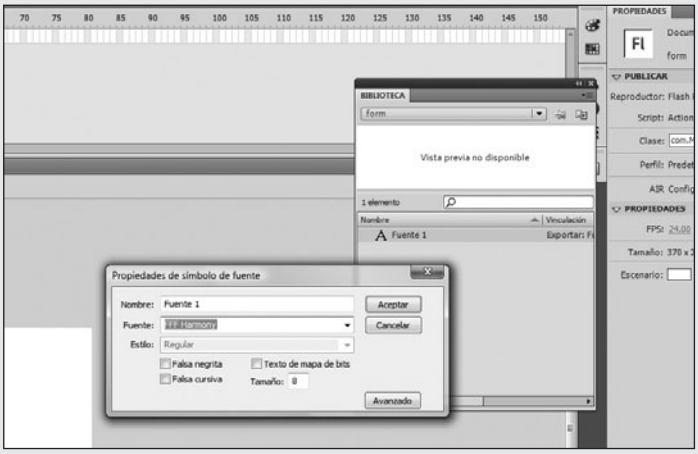
■ Embeber fuentes mediante código — PASO A PASO

1 Una vez dentro de Flash, abra la biblioteca desde **Ventana/Biblioteca**, o bien con la combinación de teclas **Ctrl+L**.

2 Presione el icono que se encuentra debajo del botón para cerrar la ventana (el que tiene una x) y seleccione, del menú que aparece, la opción **Nueva fuente...**.

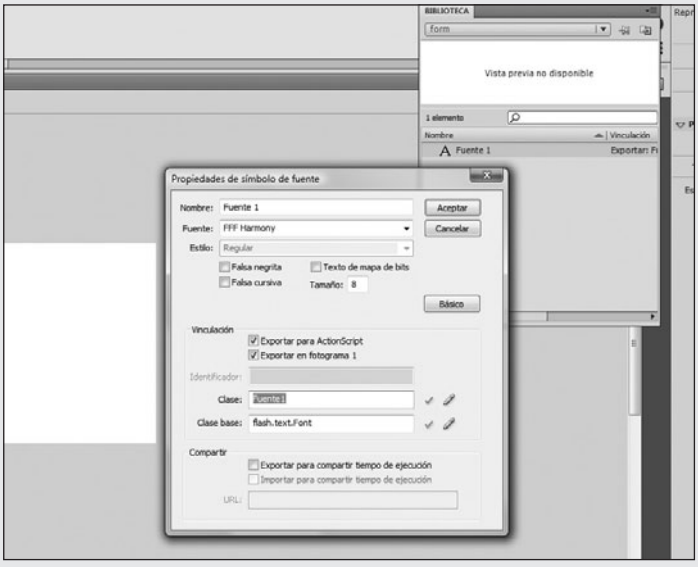


3 Una vez dentro de la ventana **Propiedades de símbolo de fuente**, seleccione, del menú expandible **Fuente**, la tipografía que desea importar y luego defina un nombre para ella. En este caso, **Fuente 1** para la fuente **FFF Harmony**.



4 Presione el botón **Avanzado** y, dentro del apartado **Vinculación**, marque la casilla **Exportar para ActionScript** de la sección **Vinculación**.

5 Defina el nombre con el que vinculará la fuente. Para este ejemplo, será **Fuente1** (sin espacio entre la cadena **Fuente** y el **1**). Éste es el nombre que utilizaremos al referirnos a esta fuente que acabamos de importar dentro de nuestro código.



Luego del procedimiento anterior, veremos que una vez que embebimos la fuente, ésta aparecerá en la biblioteca con su respectivo **nombre** y la **vinculación**, que es lo que necesitamos para utilizarla en nuestro código.

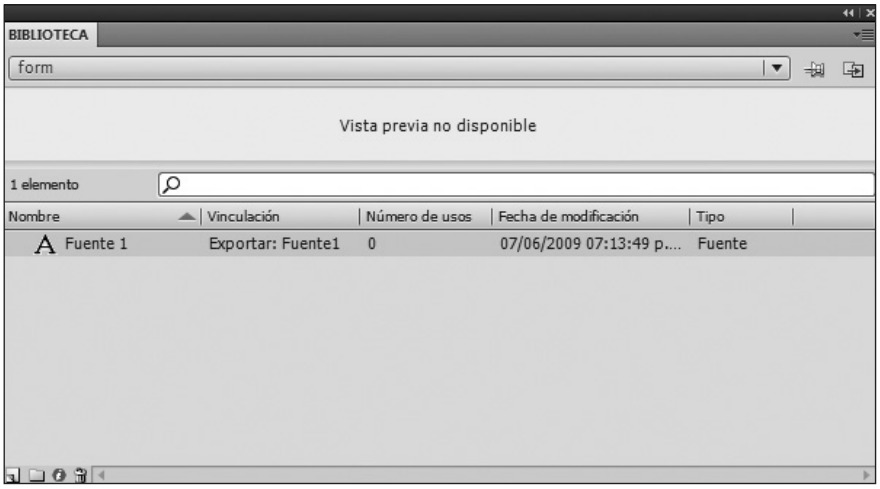


***Figura 5.** El **nombre** de la fuente es aquél con el que la identificamos y la **vinculación** es la cadena de texto que utilizamos para referirnos a una fuente dentro de nuestro código.*

Una de las ventajas al trabajar con fuentes embebidas es que al texto se le aplica un **anti-alias**, mejorando su apariencia y su legibilidad. Por otro lado, un campo de texto con fuentes embebidas puede ser rotado utilizando la propiedad **rotation** y también podemos aplicarle transparencias (propiedad **alpha**). A pesar de estas ventajas, debemos considerar que al desarrollar nuestras aplicaciones con fuentes embebidas se incrementa el tamaño final de nuestras aplicaciones. Por este motivo, debemos ser cuidadosos y medidos a la hora de embeber fuentes y lograr un justo equilibrio entre el uso de fuentes y el peso final del archivo.
Una vez que embebimos la fuente, para usarla debemos crear un **formato** para aplicarlo a nuestros campos de texto. En este formato tenemos que definir varios parámetros que, como su nombre lo indica, hacen al formato del texto: fuente que se va a utilizar, el tamaño que tendrá y el tipo de alineación, entre otras opciones que veremos a continuación.

Formato de texto: TextFormat

La clase **`TextFormat`** se emplea para aplicar un determinado formato a nuestros campos de texto. Para crear un formato, debemos hacerlo por medio del constructor **`new TextFormat()`**. La tabla que aparece en la próxima página muestra todas las propiedades de la clase **`TextFormat`**:

PROPIEDAD	VALOR POR DEFECTO
align	"left"
blockIndent	0
bold	false
bullet	false
color	0x000000
font	"Times New Roman"
Indent	0
italic	false
kerning	false
leading	0
leftMargin	0
letterSpacing	0
rightMargin	0
size	12
tabStops	[]
target	""
underline	false
url	""

***Tabla 1.** Listado de las propiedades que nos brinda la clase TextFormat para darle formato a un campo de texto.*

Si analizamos nuestro código, veremos que contamos con la función **crearFormato()**, y que en ella establecemos algunas características:

```
private function crearFormato():void{
     var font:Fuente1 =new Fuente1();
     formato = new TextFormat();
     formato.font= font.fontName;
     formato.size=8;
     formato.align='left';
}
```

¿Cómo crear un formato?

En primer lugar, asignamos una fuente para nuestro formato. Páginas atrás vimos de qué manera embeber una fuente para utilizarla en campos de texto creados mediante código. Ahora bien, ésta es la manera adecuada de, una vez embebida, definirla como la fuente de un determinado formato. Para eso, primero creamos la instancia de la clase **Fuente1**, que definimos al embeber la fuente. Veamos esta línea a continuación:

```
var font:Fuente1 =new Fuente1();
```

Luego, se la asignamos a nuestro formato:

```
formato = new TextFormat();
formato.font= font.fontName;
```

Como vemos en el código, también hicimos uso de otras propiedades. Por medio de la propiedad **size**, definimos el tamaño:

```
formato.size=8;
```

Y además determinamos su alineación por medio de la propiedad **align**:

```
formato.align='left';
```

Cuando establecimos el formato, se lo aplicamos a nuestro campo de texto. Primero usamos la propiedad **embedFonts**, para definir si el campo utilizará fuentes embebidas:

```
nombreEst.embedFonts = true;
```

¿Cómo aplicar un formato?

Tenemos dos maneras de aplicar un formato a un campo de texto: por medio de la propiedad **defaultTextFormat** o a través del método **setTextFormat()**. La diferencia es que el método **setTextFormat()** solamente aplica cambios a textos que ya han sido creados y se están mostrando en el campo de texto, mientras que la propiedad **defaultTextFormat** especifica de forma previa el formato que va a uti-

III CSS EN FLASH

Flash acepta hojas de estilo en cascada (**CSS**, acrónimo de **Cascading Style Sheets**). **CSS** es otra alternativa que podemos emplear para aplicar formato a nuestros campos de texto. El estilo puede ser creado dentro de Flash o también cargarse un archivo **.CSS** de forma externa. La ventaja principal de esto es que podemos editar sus contenidos desde afuera de la película.

lizar nuestro campo. Por este motivo, si utilizamos la propiedad **defaultTextFormat**, debemos hacerlo antes de asignar un texto a nuestro campo, y si usamos el método **setTextFormat()** tenemos que hacerlo después de haberlo asignado. Para nuestro ejemplo, utilizaremos la propiedad **defaultTextFormat**:

```
nombreEst.defaultTextFormat = formato;
```

Hacer más usable un formulario

Si bien hemos visto los conceptos básicos para generar los campos de texto de un formulario de contacto, hay varias maneras de que éste sea más fácil de utilizar y represente un menor esfuerzo para el usuario. Se trata de una serie de cuestiones que hacen a la **usabilidad** y vale la pena que las veamos, ya que Flash nos provee de varias ventajas en este aspecto.

Foco

El foco es la indicación de que un elemento del escenario está **seleccionado** y de que es el **target** para la interacción del teclado o de nuestro mouse. Esto es realmente útil para el manejo de campos de texto y formularios. Lo que nos interesa es asignar automáticamente un foco al campo de texto por el que queremos que nuestro usuario comience a escribir, evitándole la tarea de hacer foco manualmente sobre ese campo. Para lograrlo, debemos utilizar la propiedad **focus** del escenario (**stage**):

```
stage.focus = nombreInput;
```

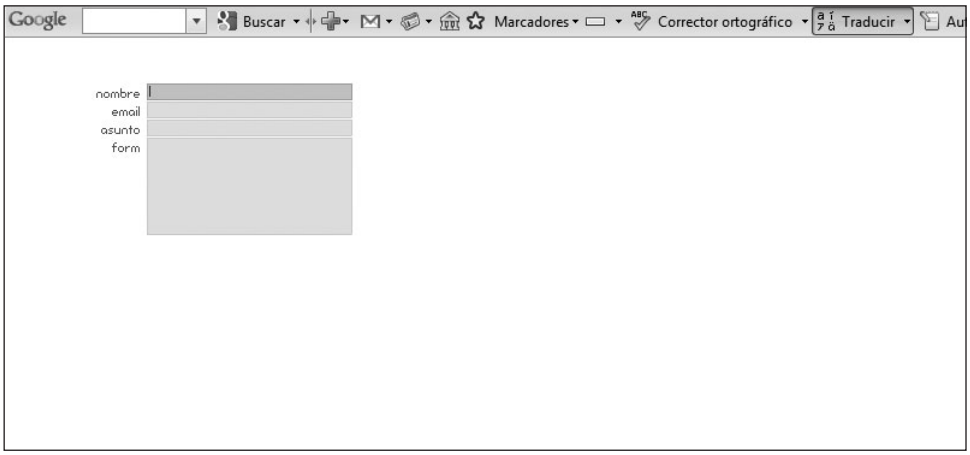


Figura 6. *Al probar la película embebida dentro de un HTML, el foco aparece en el campo al que se lo asignamos por medio de stage.focus.*

Orden de tabulación

Por medio de la propiedad **tabIndex** podemos definir el orden de tabulación. Si agregamos el índice de tabulación a cada uno de los campos del tipo **Introducción de texto** con los que contamos, veremos que al ejecutar la película y presionar la tecla **TAB**, el foco sobre los campos de texto irá cambiando en el orden que lo especificamos en nuestro código. Veamos un ejemplo:

```
nombreInput.tabIndex = 1;
emailInput.tabIndex = 2;
asuntoInput.tabIndex = 3;
formInput.tabIndex = 4;
```

Para poder definir un índice de tabulación sobre un campo de texto, se debe establecer su propiedad **tabEnabled** como **true**. No lo hemos hecho en nuestro ejemplo ya que en los campos del tipo **Introducción de texto**, el valor por defecto de esta propiedad es **true**. No sucede lo mismo con los **campos dinámicos**, a los que deberemos asignarles el valor **true** si queremos establecer un orden de tabulación sobre ellos.

Detección de foco y fuera de foco: métodos FOCUS_IN y FOCUS_OUT

Aunque estos eventos ya existían en las versiones anteriores del lenguaje, vale la pena que veamos cómo emplearlos en ActionScript 3.0. Su función es simple: por medio de los eventos **FOCUS_IN** y **FOCUS_OUT**, podemos detectar cuando el usuario hace foco o deja de hacerlo sobre un determinado campo. Esto es de enorme utilidad y nos ayuda a optimizar nuestros formularios.

Si detectamos cuando un usuario deja de hacer foco, podemos controlar los campos en tiempo de ejecución. Supongamos que el usuario tiene que ingresar un nombre de usuario, un nombre para un dominio o cualquier valor que requiera una verificación de su disponibilidad; al hacerlo y salir del campo, podemos consultar una base de datos para saber si ese nombre de usuario ya se utilizó o si se encuentra disponible, e informarle de esto al usuario antes de que envíe el formulario. Esto optimiza en gran medida nuestras aplicaciones, ya que va guiando al usuario en tiempo real y evita la molesta tarea de tener que enviar un formulario para que éste nos informe acerca de los campos incompletos, de los errores que cometimos o de la no disponibilidad de algún dominio o nombre de usuario. Para detectar estos eventos, debemos asignarlos a los campos de texto. En nuestro caso, los aplicamos a todos los campos del tipo **Introducción de texto**:

```
emailInput.addEventListener(FocusEvent.FOCUS_IN, onFocus);
emailInput.addEventListener(FocusEvent.FOCUS_OUT, killFocus);
```

Luego, definimos las funciones para cuando se produce cada uno de los eventos.

```
private function onFocus(event:FocusEvent):void{
     event.target.backgroundColor = 0xC9C9C9;
     event.target.borderColor = 0xaaaaaa;
     event.target.textColor = 0x333333;
}
private function killFocus(event:FocusEvent):void{
     event.target.backgroundColor = 0xE1E1E1;
     event.target.borderColor = 0xD0D0D0;
     event.target.textColor = 0x666666;
}
```

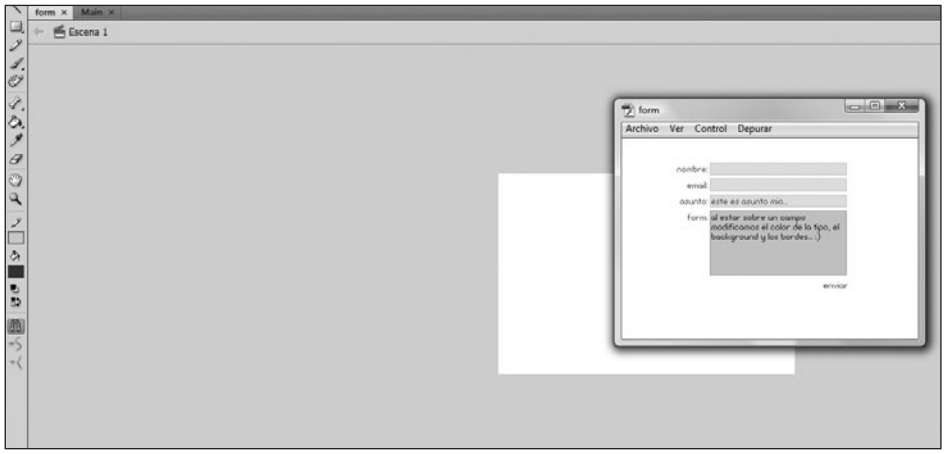


Figura 7. *Detección del evento* ***FOCUS_IN*** *sobre el campo de texto* ***formInput****, de tipo* ***Introducción de texto****. Gracias a eso, cambiamos el foco, los colores del fondo, del borde y de la tipografía del campo.*

Detección de escritura

Si bien hay algunas acciones que nos pueden ser de gran utilidad al detectar el foco o fuera de foco, hay otras que deben transcurrir mientras el usuario escribe sobre el campo. Por ejemplo, imaginemos el campo de una contraseña para el que generamos un algoritmo que se encargará de evaluar la fortaleza o debilidad de ésta. Detectando la escritura del usuario podemos informarle, en tiempo real, cómo va variando la complejidad de la contraseña a medida que la escribe, algo que resulta más interesante y útil para el usuario que si se lo informásemos una vez que terminó de escribirla, eliminó el foco y se encuentra en otro campo. O imaginemos un campo al que se le estableció una cantidad máxima de caracteres y sobre el que deseamos generar un contador que le informe al usuario cuántos caracteres le quedan para llegar al máximo permitido.

Es importante saber que la detección de la escritura corre por parte de un **evento de teclado** y no de un evento de un campo de texto, pero necesitamos del evento del campo de texto para saber cuándo el usuario está escribiendo sobre el campo en el que nos interesa detectar la escritura.
Hagamos el segundo ejemplo que dimos y generemos un contador que le indique al usuario cuántos caracteres le restan para llegar al máximo permitido. En primer lugar, definimos esta cantidad, que será de 150 caracteres:

```
private var max:uint = 150;
```

Luego, establecemos este valor como la cantidad máxima de caracteres permitidos para el campo que queremos controlar:

```
formInput.maxChars = max;
```

También necesitamos generar un campo de texto en el que mostraremos el contador de caracteres. A este campo lo llamaremos **charsDisplay**:

```
charsDisplay = new TextField();
charsDisplay.defaultTextFormat = formato;
charsDisplay.selectable = false;
charsDisplay.autoSize = TextFieldAutoSize.LEFT;
charsDisplay.textColor = 0x666666;
charsDisplay.x = 110;
charsDisplay.y = 160;
mainContainer.addChild(charsDisplay);
```

Ahora bien, debemos saber cuándo se está escribiendo sobre el campo **formInput**, que es el que nos interesa. Para esto, definimos un nombre para nuestro campo:

```
formInput.name = "form";
```

Así, al detectarse el evento **FOCUS_IN** y ejecutarse la función **onFocus()**, por medio de un condicional averiguamos sobre qué campo se está haciendo foco. En caso de ser el campo llamado **"form"**, agregamos un **listener** para detectar el evento **KEY_UP** del teclado o, en caso contrario, eliminamos el listener para que no siga ejecutándose cuando estamos escribiendo sobre otros campos.

```
private function onFocus(event:FocusEvent):void{
     if(event.target.name == “form” && !isOverForm){
            isOverForm = true;
            stage.addEventListener(KeyboardEvent.KEY_UP, checkLength, fal
se, 0, true);
     }else if(isOverForm){
            isOverForm = false;
            stage.removeEventListener(KeyboardEvent.KEY_UP, checkLength);
     }
     event.target.backgroundColor = 0xC9C9C9;
     event.target.borderColor = 0xaaaaaa;
     event.target.textColor = 0x333333;
}
```

Por último, veamos la función **checkLength**, que informa en pantalla la cantidad restante de caracteres y la cantidad total:

```
private function checkLength(event:KeyboardEvent):void{
     var rest:uint = max - formInput.length;
     charsDisplay.text = rest.toString() + “ / “ + max.toString();
}
```

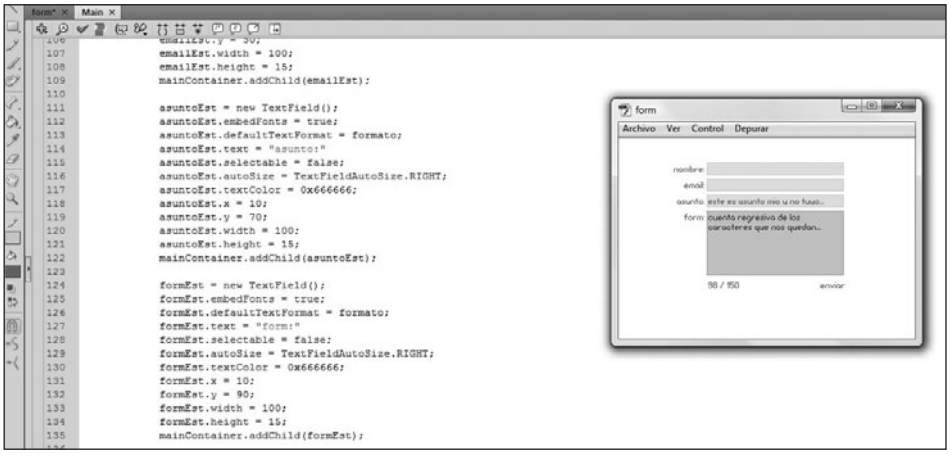


Figura 8. *Formulario de contacto con el conteo de caracteres del campo* ***formInput****. Debajo del último campo de texto vemos los caracteres restantes y los permitidos.*

Como vimos, hacer un buen uso de las propiedades y de los métodos de los campos de texto y de sus formatos nos permite generar aplicaciones más usables y contribuir a

una mejor experiencia para el usuario. A partir de ahora, veremos cómo controlar y validar los campos de texto para luego integrar nuestro formulario con PHP.

CONTROL DE VARIABLES Y ENVÍO DE FORMULARIO

Dedicaremos esta segunda parte del capítulo a dos cuestiones. Por un lado, veremos de qué manera verificar que los valores que ingresan los usuarios en el formulario sean correctos (o al menos que cumplan con los requisitos que indiquemos), y por otro, en caso de que estos valores cumplan con los requisitos, enviaremos las variables a un lenguaje del lado del servidor para que éste los haga llegar a una dirección de e-mail.

Verificación de datos en Flash

A continuación crearemos un ejemplo que nos permitirá verificar los datos que han sido ingresados, corroborar que sean correctos y posteriormente enviarlos a PHP. Antes de comenzar, debemos agregar algunos contenidos a nuestro formulario. En primer lugar, generaremos un botón que utilizaremos para hacer el envío de las variables:

```
enviar = new Sprite();
enviar.addEventListener(MouseEvent.MOUSE_DOWN, checkVars, false, 0, true);
enviar.mouseChildren = false;
enviar.buttonMode = true;

enviarEst = new TextField();
enviarEst.x = 1;
enviarEst.defaultTextFormat = formato;
enviarEst.text = “enviar”;
enviarEst.selectable = false;
enviarEst.autoSize = TextFieldAutoSize.LEFT;
enviarEst.textColor = 0x666666;
enviar.addChild(enviarEst);

enviar.graphics.beginFill(0xcccccc, 0);
enviar.graphics.drawRect(0, 0, enviarEst.textWidth, 13);
enviar.graphics.endFill();
mainContainer.addChild(enviar);
enviar.x = 280 - enviarEst.textWidth;
enviar.y = 175;
```

Dentro del botón que acabamos de crear, generamos un campo de texto que contendrá la cadena de texto **enviar**. En este bloque de código hay dos conceptos nuevos. Por un lado, encontramos el uso de la propiedad **mouseChildren**. Esta propiedad se encarga de determinar si los elementos que se encuentran dentro del contenedor al que se le aplica la propiedad están habilitados para el mouse. Su valor por defecto es **true**, por lo que al hacer clic en el contenedor, Flash hace que el clic se produzca sobre los elementos que se encuentran **dentro del contenedor** y no sobre el contenedor. Para evitar que suceda esto, definimos esta propiedad **mouseChildren** en **false** para el Sprite **enviar**, con la finalidad de que al detectarse el clic éste se produzca sobre ese Sprite y no sobre el campo de texto **enviarEst** que se encuentra dentro de él.
Luego tenemos la propiedad **buttonMode**, que se encarga de definir si nuestro botón se comportará como tal o no. Al asignarle el valor **true**, cuando nos posicionamos sobre él se activará el cursor que nos indica que estamos sobre un botón (el cursor con forma de mano).
Otro contenido que nos hace falta para que nuestro formulario sea funcional es la creación de un campo de texto que nos mantenga al tanto de lo que está sucediendo, que nos informe cuando hay un campo incompleto, cuando se están enviando las variables y si se enviaron exitosamente o si falló el envío. En nuestro ejemplo, a este campo lo llamaremos **displayStatus**.
Ahora bien, ya tenemos todos los contenidos que necesitamos para hacer funcionar nuestro formulario. Integremos todo y veamos de qué manera hacer llegar las variables al servidor (PHP en este caso). Recordemos que al sprite **enviar** le asignamos un **Event Listener** que se activará al liberar el clic del mouse:

```
enviar.addEventListener(MouseEvent.MOUSE_DOWN, checkVars, false, 0, true);
```

Como veremos, antes de enviar las variables es necesario que chequeemos que sus valores sean correctos. Por este motivo, llamamos a la función **checkVars**.

Validar datos: expresiones regulares

Las expresiones regulares son nuevas en el mundo Flash y son una de las características más interesantes y potentes del lenguaje. Básicamente, una expresión regular es un **patrón** que se emplea para buscar y manipular texto coincidente en cadenas, y su principal propósito es comprobar que los valores de determinados campos se ajustan a un patrón específico. Para verlo en nuestro ejemplo, analicemos el campo que asignamos a nuestro formulario para que el usuario indique su e-mail: sabemos que una dirección de correo electrónico cuenta con una serie de patrones que se deben cumplir para que sea válida:

- La cadena de texto no debe contener espacios.
- La cadena de texto debe contener una única arroba.
- Debe haber un nombre para la dirección antes de la arroba.
- Debe haber un servidor, ubicado después de la arroba.
- Debe haber un dominio (.com, .net, etcétera) que puede incluir o no subdominios (.com.ar, .net.ar, etcétera).
- Hay caracteres permitidos para una dirección de e-mail y otros que no lo son.

Todas estas restricciones podríamos controlarlas por medio de **condicionales**, pero imaginemos la cantidad de líneas que necesitaríamos para corroborar y validar todos estos puntos. Escrito en una expresión regular, todos los requisitos que acabamos de mencionar están explícitos en la siguiente línea:

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

Las **expresiones regulares** son un tema complejo. Comprender su escritura y lectura implica el conocimiento de **metacaracteres**, **metasecuencias**, **indicadores** y **métodos**. Esto hace dificultoso abordar el tema en tan sólo una sección de libro. Debido a su complejidad, no analizaremos la sintaxis pero sí es importante que conozcamos cómo utilizarla y cuáles son sus propósitos. Aunque no veremos de qué modo crearlas, podemos mencionar que utilizando tan sólo una línea de código es posible ahorrar una serie de procedimientos que solían llevarse a cabo por medio de condicionales para verificar cadenas de texto. Podemos formular expresiones regulares de dos maneras. Por medio del empleo de la barra diagonal para delimitar la expresión regular:

```
var miRegExp:RegExp = /mi expresión regular/;
```

O por medio del constructor **new**:

```
var miRegExp:RegExp = new RegExp("mi expresión regular");
```

III OTROS USOS DE LAS EXPRESIONES REGULARES

Si bien en nuestro ejemplo empleamos expresiones regulares para verificar la validez de una dirección de e-mail y lograr que nuestros otros campos no estén vacios, a las Expresiones Regulares se les suele dar diversos usos, como la validación de códigos postales, de teléfonos, de números celulares y de documentos personales, entre otras cosas.

En nuestro caso utilizaremos dos expresiones: una para controlar que los campos no estén vacíos y otra para verificar la dirección de e-mail:

```
var fieldsRegExp:RegExp = /\S/;
var emailRegExp:RegExp = /^([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-
Z]+[.])+[a-zA-Z]{2,6}$/;
```

El método **test()** se encarga de comprobar si existe una coincidencia en la cadena dada, devolviendo los valores booleanos **true** o **false**. Así, verificamos todos los campos del formulario y, en caso de no cumplirse la condición, le informamos al usuario:

```
if(!fieldsRegExp.test(nombreInput.text)){
     nombreInput.backgroundColor = 0x990000;
     nombreInput.textColor = 0xffffff
     displayStatus.text = "el campo nombre es obligatorio";
```

Aplicaremos el mismo concepto a todos los campos: en caso de que no cumpla con los requisitos, cambiaremos su color de fondo, el color de la tipografía y usaremos el campo de texto **displayStatus**, creado para alertar al usuario que el campo es obligatorio.

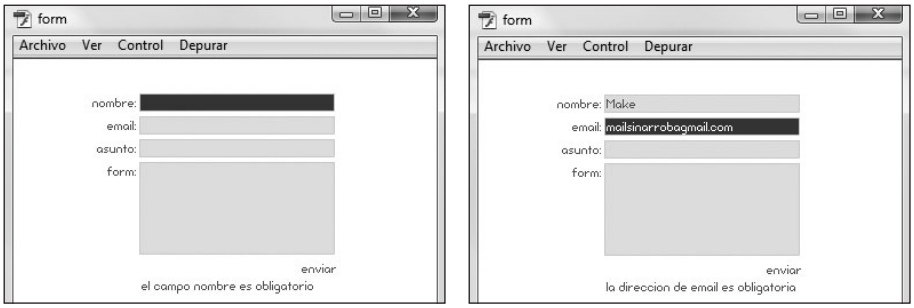


Figura 9. *Al detectar un campo incompleto, modificamos el fondo para alertar al usuario.*

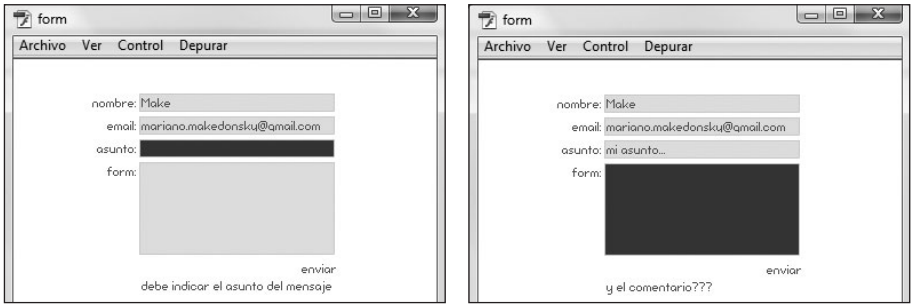


Figura 10. *A su vez, indicamos el campo incompleto en una variable.*

Enviar variables al servidor

Una vez que hemos corroborado que los campos de texto cumplen con los requisitos, debemos enviar los valores de esas variables al servidor. Para esto tenemos que usar una serie de clases que ya hemos utilizado en el capítulo anterior y otras que son nuevas. En primer lugar, veamos las clases que debemos importar para enviar variables al servidor:

```
import flash.net.URLVariables;
import flash.net.URLRequestMethod;
import flash.net.URLRequest;
import flash.net.URLLoader;
```

Como dijimos, varias de ellas ya las vimos en el capítulo anterior, pero otras son nuevas; hagamos un repaso para afianzar conocimientos.

- **URLVariables**: esta clase no la hemos empleado hasta aquí. Nos permite definir las variables que enviaremos al servidor y sus valores. Haremos uso de ella siempre que necesitemos enviar variables al servidor. Una vez creado el constructor de la clase **URLVariables**, le asignamos la cantidad de variables que necesitemos enviar. Podemos observar que a las variables creadas dentro del objeto (**nombre**, **email**, **asunto** y **form**) les asignamos los valores que se indicaron en el formulario (**nombreInput**, **emailInput**, **asuntoInput**, **formInput**). Veamos nuestro ejemplo:

```
var variables:URLVariables = new URLVariables();
variables.nombre = nombreInput.text;
variables.email = emailInput.text;
variables.asunto = asuntoInput.text;
variables.form = formInput.text;
```

- **URLRequestMethod**: nos permite especificar un valor para el tipo de método que debe emplear **URLRequest** al enviar datos a un servidor: **POST** o **GET**.
- **URLRequest**: esta clase reúne toda la información que utilizaremos para una solicitud HTTP. Veamos un ejemplo:

```
var request:URLRequest = new URLRequest();
request.url = "http://localhost/formPHP/mail.php";
request.method = URLRequestMethod.POST;
request.data = variables;
```

En nuestro caso, empleamos las siguientes propiedades: **url** es la URL a la cual enviaremos las variables: el archivo **mail.php**. La propiedad **method** define de qué manera enviaremos la información al servidor: **URLRequestMethod.POST**. Vimos la finalidad de la clase **URLRequestMethod** anteriormente. La utilizaremos para que el envío sea por medio de **POST**. La propiedad **data** contiene las variables que enviaremos. Como vemos, le asignamos el objeto **variables** que creamos anteriormente:

```
request.data = variables;
```

- **URLLoader**: se ocupa de descargar datos desde una URL como texto, sean estos datos binarios o variables. En nuestro ejemplo, por medio de la clase **URLLoader** recibiremos una respuesta de PHP a fin de saber si el mensaje se envió correctamente o no.

```
var loader:URLLoader=new URLLoader();
loader.addEventListener(Event.COMPLETE,loadComplete);
loader.load(request);
function loadComplete(event:Event) {
     var loader:URLLoader= URLLoader(event.target);
     if(loader.data.r==0){
            displayStatus.text = “error al eviar...”;
     }else{
            displayStatus.text = “email enviado! :)”;
     }
}
```

Recibir variables en PHP desde Flash

Recordemos que no es PHP la finalidad de este libro y, por ese motivo, acotaremos la sintaxis al código básico e indispensable para que funcione correctamente nuestro ejemplo. En el archivo **mail.php** necesitamos código para recibir nuestras

INSTALACIÓN DE PHP

Ya hemos conocido la información de la instalación de **PHP** en el capítulo anterior. Vale recordar que para hacer uso de estos ejemplos, debemos contar con un hosting que soporte PHP o bien haber instalado previamente el servicio en modo local. En ese mismo capítulo aparece un listado con los distintos servidores **WAMP** y **MAMP** para hacer correr el servidor Apache localmente.

variables de Flash en PHP y luego enviarlas a una dirección de e-mail, utilizando la función **mail()**. A continuación veremos la sintaxis de este archivo:

```
<?php

//variables que recibimos desde Flash en POST:

$nombre = $_POST['nombre'];
$email = $_POST['email'];
$asunto = $_POST['asunto'];
$form = $_POST['form'];

//Variables para la función mail:

$to="me@localhost";
$subject= $asunto;
$header="from: " . $email . "<". $nombre . ">";
$message= $form;
$sentmail = mail($to,$subject,$message,$header);

if($sentmail){
echo "r=1";
}
else {
echo "r=0";
}

?>
```

En primer lugar, creamos cuatro variables en PHP, a las que les asignaremos, como valores, las cuatro variables que enviamos desde Flash:

```
$nombre = $_POST['nombre'];
$email = $_POST['email'];
$asunto = $_POST['asunto'];
$form = $_POST['form'];
```

Luego, en la parte más importante de nuestro código, definimos las variables que enviaremos por medio de la función **mail()**. Veamos esto a continuación:

```
$to=”me@localhost”;
$subject= $asunto;
$header=”from: “ . $email . “<”. $nombre . “>”;
$message= $form;
$sentmail = mail($to,$subject,$message,$header);
```

A la función **mail()** debemos indicarle cuatro parámetros:

- **Dirección** de destino (**$to**).
- **Asunto** del correo (**$subject**).
- **Mensaje** del correo (**$message**).
- Los **headers** del e-mail (**$header**).

Una vez concluida la función **mail()**, por medio de un condicional comprobamos si los datos se enviaron correctamente:

```
if($sentmail){
echo “r=1”;
}
else {
echo “r=0”;
}
```

Definimos una variable llamada **r** que, en caso de que el envío haya sido exitoso, tendrá valor **1** y, en caso contrario, será **0**. Esta información la tomamos en Flash por medio de **URLLoader**, como vimos anteriormente:

```
function loadComplete(event:Event) {
var loader:URLLoader= URLLoader(event.target)
if(loader.data.r==0){
            displayStatus.text = “error al enviar...”;
     }else{
            displayStatus.text = “email enviado! :)”;
     }
}
```

De este modo, le informamos al usuario si el correo se envió de forma correcta o si hubo un error durante el envío, como vemos en las siguientes imágenes.

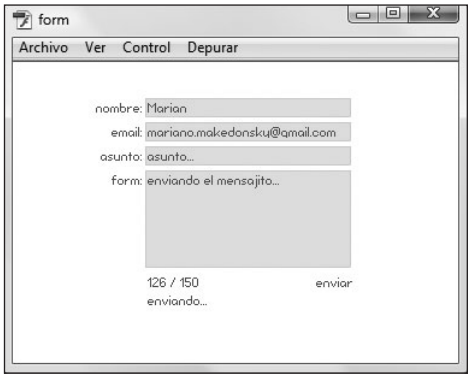
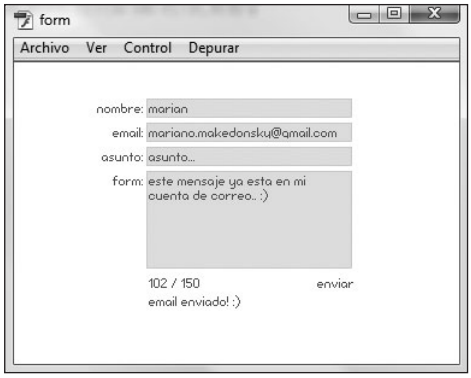


Figura 11. *Definimos el valor* ***enviando...*** *a la variable* ***displayStatus*** *para que se muestre al hacer el envío de las variables al servidor.*



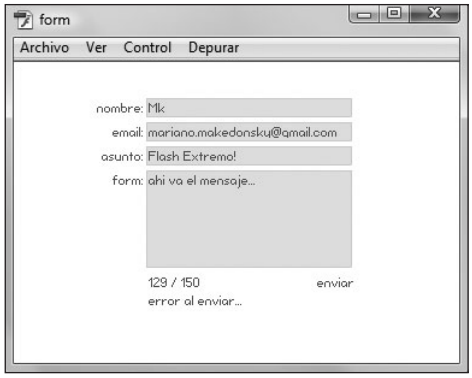


Figura 12. *Una vez que recibimos la respuesta desde PHP, el formulario nos informa si nuestro mensaje se ha podido enviar con éxito o si falló.*

... RESUMEN

El buen uso de campos de texto, tanto desde la IDE como desde las propiedades y métodos de la clase TextField, nos permite optimizar nuestros desarrollos, generando aplicaciones que aportan una mejor experiencia para el usuario. Por otro lado, la flexibilidad de Flash para la comunicación cliente/servidor nos facilita el envío y la recepción de variables, haciendo completamente funcionales nuestros desarrollos. En el próximo capítulo, nos adentraremos en el uso de imágenes, XML y APIS externas.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1) ¿Con qué tipos de texto contamos en Flash?

2) ¿Cuáles son sus diferencias y cuándo es óptimo el uso de cada uno?

3) ¿De qué manera se embeben fuentes para textos dinámicos creados desde la IDE de Flash? ¿Y para campos de texto creados mediante ActionScript?

4) ¿Cuál es la principal ventaja de emplear código para el manejo de campos de textos?

5) ¿Qué clase se utiliza para la creación de campos de texto mediante ActionScript?

6) ¿Qué propiedades se utilizan para definir el texto, el color, el ancho, el largo, la visibilidad de bordes y fondos y sus respectivos colores en campos de textos creados mediante la clase **TextField**?

7) ¿Qué clase utilizamos para asignarle un formato a un campo de texto?

8) ¿Qué propiedad aplicamos para que un campo sea del tipo **Introducción de texto**?

9) ¿Qué evento se utiliza para detectar cuando un usuario hace foco o quita el foco de un campo de texto?

10) ¿Qué son las expresiones regulares? ¿Para qué se utilizan?

EJERCICIOS PRÁCTICOS

1) Cree su propio formulario de contacto con los campos que le resulten convenientes. Puede hacerlo desde la IDE de Flash o mediante ActionScript.

2) Haga que, al abrir su película, el foco se sitúe automáticamente sobre el primer campo del formulario.

3) Defina un orden lógico de tabulación para sus campos que sean de tipo Introducción de texto.

4) Restrinja la cantidad y el tipo de caracteres en caso de contar con campos que así lo requieran.

5) Modifique las propiedades backgroundColor, borderColor y textColor cada vez que el usuario haga foco en cada campo de texto.

6) Cree dinámicamente o agregue un botón para el envío de las variables de su formulario.

7) Previo al envío, verifique que los campos obligatorios del formulario estén completos.

8) En base a lo hecho en el punto anterior, advierta al usuario en caso de que haya campos obligatorios incompletos o proceda al envío de las variables.

Imágenes en Flash

En este capítulo veremos el uso de imágenes en Flash, analizaremos las distintas posibilidades para utilizarlas y veremos de qué manera optimizar tanto los mapas de bits como nuestras películas. Además, nos adentraremos en la integración de Flash con APIs y aprenderemos cómo implementar desarrollos con los servicios de Flickr para generar un buscador de imágenes.

INTRODUCCIÓN AL USO DE IMÁGENES

La evolución hacia la Web 2.0, por un lado, y el constante desarrollo de Flash, por el otro, han hecho que el actual modo de trabajar y pensar las imágenes difiera de la manera en la que se hizo en un comienzo. Para empezar, veremos las distintas modalidades de utilizar los **bitmaps**, las ventajas y desventajas para cada caso y, finalmente, crearemos nuestra propia galería con la mayor optimización posible.

Imágenes en Flash

Flash nos permite utilizar imágenes de dos modos: como imágenes internas y como imágenes externas. Las **imágenes internas**, como su nombre lo indica, se ubicarán dentro de nuestro archivo .FLA. Podemos importar las imágenes a la **biblioteca** de nuestro archivo o al **escenario**. Para hacerlo, debemos ir a **Archivo**, seleccionar del menú la opción **Importar** y luego elegir entre **Importar a escenario...** o **Importar a biblioteca...**.

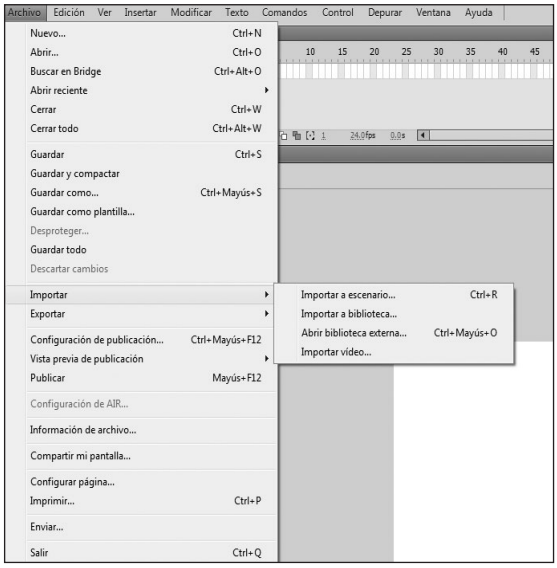


Figura 1. *También podemos importar imágenes al escenario presionando* ***CTRL+R****.*

Esta modalidad puede ser útil en el caso de que necesitemos sólo una pequeña cantidad de imágenes en nuestro desarrollo; por ejemplo, alguna imagen de fondo para el proyecto que estemos llevando a cabo o una imagen chica dentro de la pantalla.
Como beneficio de incorporar las imágenes dentro del archivo .FLA, podemos mencionar la **disponibilidad de archivos**. La única **ventaja** que implica tener las imágenes dentro de nuestra película es que una vez cargada, disponemos de ellas inmediatamente. Esto puede ser útil en aplicaciones locales donde los pesos de los archivos no son un factor determinante.

Entre las **desventajas** de agregar las imágenes dentro del archivo .FLA, es posible mencionar las que veremos a continuación:

- **Dificultosa actualización de contenidos**: en el caso de tener que actualizar los contenidos, debemos abrir el archivo .FLA cada vez que esta tarea se requiera, editar los contenidos y luego volver a exportar la película.
- **Peso de nuestra película**: el problema que surge a la hora de emplear imágenes dentro de Flash es que el archivo .SWF resultante irá incrementando gradualmente su peso a medida que incluyamos más imágenes. Imaginemos una pequeña galería que contenga tan sólo diez imágenes cuyos pesos ronden los 100 kb; tendríamos 1 Mb dentro de nuestra película, que el usuario tendrá que descargar para poder visualizar la galería.

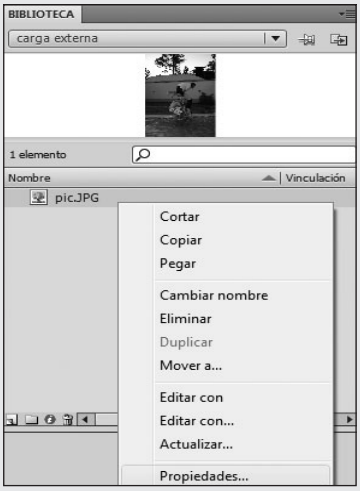
Compresión de imágenes importadas

Al importar imágenes a Flash, podemos modificar su calidad a fin de reducir su peso y el de la película. Este procedimiento disminuirá considerablemente el tamaño del archivo, ya que a mayor compresión, menor peso de imagen y por ende menor peso final de la película. Veamos cómo lograrlo.

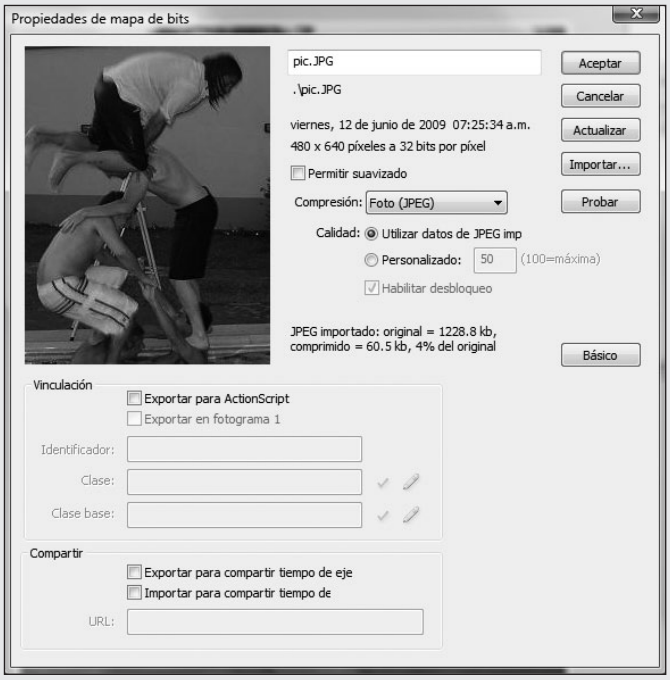
■ Reducir la calidad de una imagen — PASO A PASO

1 Una vez dentro de Flash, abra la biblioteca desde **Ventana/Biblioteca**, o bien presionando la combinación de teclas **CTRL+L**.

2 En el interior de la biblioteca, haga clic con el botón derecho sobre la imagen a la que desea modificarle la calidad y presione **Propiedades...**.



3 En la ventana **Propiedades de mapa de bits**, deje seleccionada la opción **Foto (JPEG)** dentro de la **Compresión**.



4 A continuación, establezca la opción **Personalizado** para la calidad de la imagen e ingrese un valor entre 1 y 100 para determinar la definición que tendrá la imagen. 100 es la máxima calidad y 1 la mínima. Para terminar, presione el botón **Aceptar** para que se apliquen las configuraciones elegidas.

Imágenes externas

Flash nos permite cargar contenidos externos. En la siguiente página, podemos ver todo el código que necesitamos para importar una imagen:

III TAMAÑO DE IMÁGENES, FORMATO JPG

La ventaja del formato JPEG es, justamente, la posibilidad de comprimir las imágenes y definir el grado de compresión. En base a esto, es importante hacer un justo balance entre el peso final del archivo y su calidad. A menor compresión, mayores serán la calidad y el tamaño del archivo y a menor compresión, obtendremos imágenes de menor tamaño, pero de baja calidad.

```
var myRequest:URLRequest = new URLRequest();
myRequest.url = "pic.jpg";

var myLoader:Loader = new Loader();
myLoader.x = myLoader.y = 5;
myLoader.load(myRequest);

addChild(myLoader);
```

En el código hay una clase de la que no hemos hecho uso hasta ahora: la clase **Loader()**. Ésta se utiliza para cargar imágenes externas (**JPG**, **PNG** o **GIF**) y también nos permite cargar películas **SWF**. Para cargar una imagen, en primer lugar debemos crear una **instancia** de la clase **Loader** y luego, desde el constructor, utilizamos el método **load()** para indicar el archivo que queremos incorporar:

```
var myLoader:Loader = new Loader();
myLoader.load(myRequest);
```

Una vez que contamos con el objeto **Loader**, podemos manejarlo como cualquier elemento de nuestra lista de visualización (**Display List**):

```
myLoader.x = myLoader.y = 5;
addChild(myLoader);
```

La clase **Loader** es realmente muy completa, y la conoceremos a lo largo de este capítulo. Entre las ventajas de la carga externa de imágenes, podemos mencionar:

- **Actualización**: en base a la lectura de estructuras XML, que veremos en las próximas páginas, los contenidos de nuestra película podrán actualizarse de ma-

III ¿CÓMO EXPORTAR IMÁGENES PARA LA WEB DESDE PHOTOSHOP?

Si somos usuarios de Flash, seguramente utilicemos **Adobe Photoshop**. Desde él podemos acceder a la opción **Guardar para web o dispositivos...** en el menú **Archivo** o presionando **ALT + MAYÚS+ CTRL + S**. De este modo, se abrirá una pantalla para exportar el archivo en formato **JPG**, **PNG** o **GIF**, con distintas alternativas de calidad y compresión a fin de optimizar la imagen para la Web.

nera externa, de modo que no haya que tocar el archivo .FLA. Simplemente, añadiremos **nodos** a nuestro **XML** indicando la imagen que se va a cargar y la copiaremos en el lugar que corresponda.

- **Peso de archivo**: independientemente del número de imágenes que se vaya a mostrar, el peso final de nuestra película será siempre el mismo.

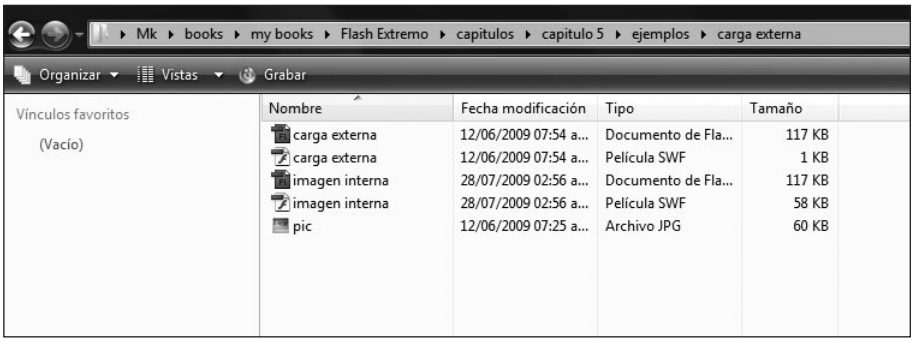


***Figura 2.** El archivo carga externa.swf no llega a **1 kb** y pesará siempre lo mismo, independientemente de la cantidad de imágenes que utilicemos. El archivo imagen interna.swf pesa **58 kb** y contiene tan sólo una imagen.*

Conclusiones

La intención de este pequeño análisis que hicimos entre los distintos tipos de contenidos (internos y externos) es notar que la flexibilidad y las alternativas de cada uno varían. El empleo de imágenes dentro de nuestro archivo .FLA es útil solamente si contamos con pocas imágenes y de poco peso, o bien para aplicaciones locales que no requieran de la descarga de la película. Si nuestro desarrollo va a tener una mayor complejidad que implica la carga de varios contenidos y la necesidad de actualizarlos, debemos pensar en cargarlos de manera externa.

CREACIÓN DE UNA GALERÍA DE IMÁGENES DINÁMICA

Lo más interesante del ejemplo que analizaremos a continuación es que no sólo abarcaremos nuevos aspectos del desarrollo en Flash, sino que pondremos en práctica todo lo que hemos aprendido hasta ahora con este libro. Si bien vimos que la carga de imágenes externas es sencilla, a la hora de desarrollar una galería no debemos importar una única imagen sino varias, por lo que necesitamos de una estructura que nos agilice el desarrollo y, posteriormente, su mantenimiento. XML será el metalenguaje que nos permitirá llevar a cabo estas tareas.

Breve introducción a XML

XML es el acrónimo de **eXtensible Markup Language** (lenguaje de marcas extensible en español), un metalenguaje desarrollado por la **W3C** (World Wide Web Consortium, **www.w3c.org**). Su principal ventaja es la de permitir el intercambio de información entre distintas plataformas de una manera muy fácil, debido a su flexibilidad. Veamos un ejemplo:

```
<miXML>
     <miNodo atributo="atributo del nodo ">contenido</miNodo>
     <miNodo atributo="otro atributo">otro contenido</miNodo>
</miXML>
```

Un documento XML está compuesto por **etiquetas**, las cuales no están predefinidas sino que sus nombres los establece el usuario, como por ejemplo **<miXML>**. Para cerrar una etiqueta, debemos anteponer una barra delante de su nombre, o bien al final del **nodo** sin indicar la etiqueta de cierre:

```
</miXML>
//o de esta otra forma:
<miNodo atributo="atributo del nodo"/>
```

Dentro de una estructura, creamos sus **nodos**. Un XML es una estructura jerárquica, y esta jerarquía es lo que hace que existan **nodos padres** o **nodos hijos**. Los primeros son aquellos que contienen nodos en su interior y los nodos hijos son aquellos que se encuentran dentro de otro nodo:

```
<nodoPadre>
     <nodoHijo></nodoHijo>
</nodoPadre>
```

III ESTRUCTURAS XML

Si bien emplearemos un archivo XML para la creación de una galería, ésta es tan sólo una de las tantas aplicaciones que podemos desarrollar implementando el metalenguaje. Es posible utilizar XML para cualquier tipo de aplicación que requiera la carga de contenidos dinámicos, sean estos **reproductores de MP3**, de **video**, **gestores de noticias**, **etcétera**.

A su vez, los nodos pueden tener **atributos**:

```
<miXML>
     <miNodo atributo=”atributo del nodo” ></miNodo>
</miXML>
```

Y contenidos:

```
<miXML>
     <miNodo atributo=”atributo del nodo “>contenido</miNodo>
</miXML>
```

Conociendo su sintaxis básica, es posible crear nuestras propias estructuras, que luego podemos cargar desde Flash e interpretar.

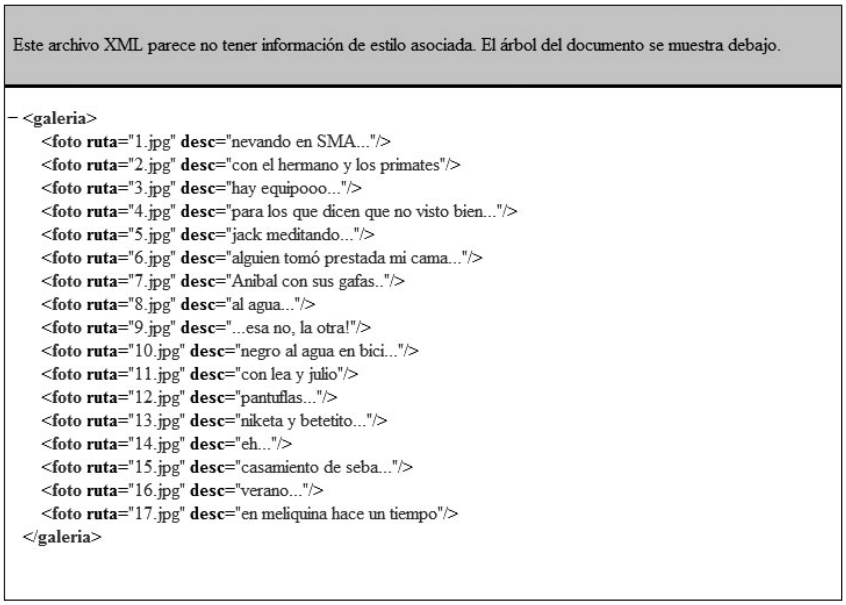


***Figura 3.** Un ejemplo de la estructura XML que usaremos para nuestra galería. Cada nodo contiene dos **atributos**: uno para la **ruta** de la imagen y otro para su **descripción**.*

Estas estructuras que generamos y podemos importar desde Flash nos facilitan el modo de llevar a cabo nuestros desarrollos con una cantidad de información considerable y compleja. Volcado a nuestro ejemplo, crearemos una estructura dentro de la cual indicaremos el nombre de las imágenes y una descripción de cada una de ellas:

```
<foto ruta="5.jpg" desc="jack meditando..."/>
<foto ruta="6.jpg" desc="alguien tomó prestada mi cama..."/>
```

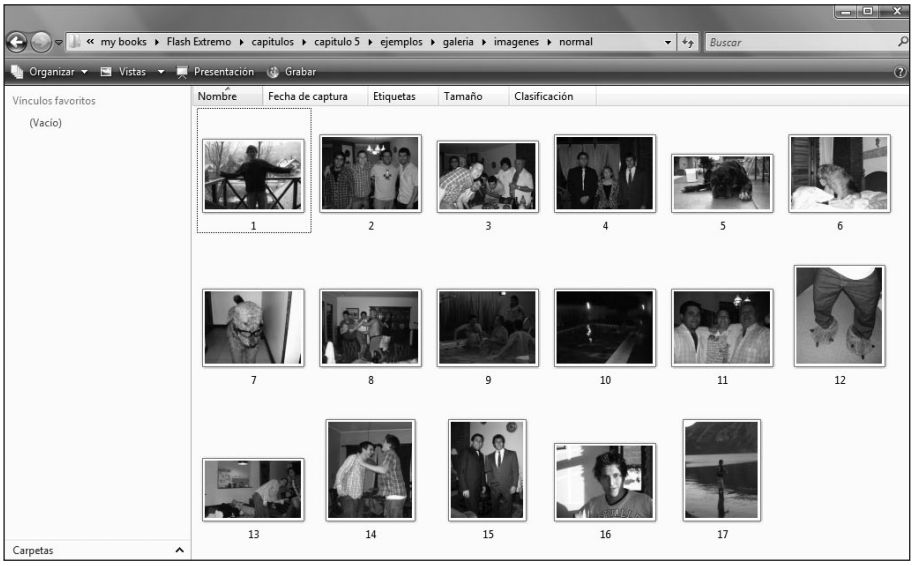


Figura 4. *El* ***atributo*** *ruta de cada nodo del* ***XML*** *indica el nombre de cada una de las imágenes, que se ubican dentro de las carpetas* ***imagenes/normal*** *e* ***imagenes/thumbs.***

XML en FLASH

En este ejemplo puntual utilizaremos XML para contar con una estructura que nos facilite la carga externa de las imágenes, por lo que necesitamos ubicarlas dentro de una ruta a la que accedamos desde Flash. Para esto, crearemos una **carpeta** y la nombraremos **imagenes**, y dentro de ella generaremos otras dos carpetas a las que llamaremos **normal** y **thumbs**. En la carpeta **normal** colocaremos las imágenes a tamaño real y dentro de **thumbs** las correspondientes miniaturas. Denominaremos del mismo modo a la imagen en miniatura y a la imagen original:

imagenes/normal/1.jpg
imagenes/thumbs/1.jpg
imagenes/normal/2.jpg
imagenes/thumbs/2.jpg
Etcétera.

Llamaremos **data.xml** a la estructura XML con la que trabajaremos. Nuestro desarrollo en Flash, al igual que para el resto del libro, lo podemos llevar a cabo desde la IDE de Flash o creando una clase. Si bien este ejemplo está basado en la clase

galeria/com/MainGallery.as, por una cuestión de practicidad simplemente nos abocaremos al código que hace al funcionamiento en relación con el manejo de imágenes.

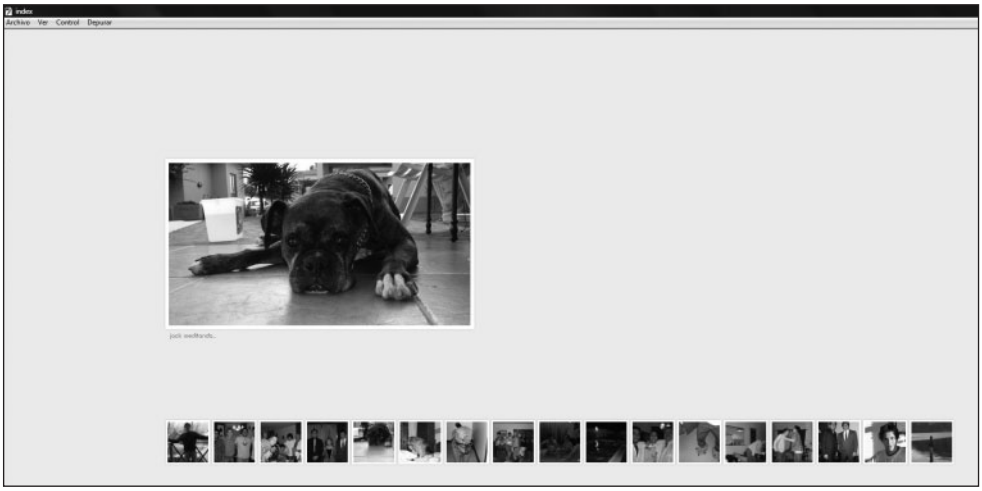


Figura 5. *Resultado final de la galería que desarrollaremos a lo largo de este capítulo.*

Ahora nos abocaremos a las nuevas cuestiones de este capítulo. Veamos:

```
public function MainGallery():void {
     crearFormato();
     crearContenidos();
     cargarXML();
}
```

La función **crearFormato()** se encargará de generar el formato que aplicaremos a los campos de texto que utilizaremos, tanto para los **preloaders** de las miniaturas (**thumbnails**) como para el epígrafe de cada foto. Por su parte, la función **crearContenidos()**, como su nombre lo indica, generará todos los **sprites**, **máscaras** y **loaders** que debamos utilizar a lo largo del desarrollo.

ENMASCARAR CONTENIDOS: PROPIEDAD MASK

Para asignar máscaras a los contenidos, debemos hacerlo por medio de la propiedad **mask**. El **display object** que origina la llamada es enmascarado mediante el objeto que se especifique: **objetoAEnmascarar.mask = objetoMascara;**. Si deseamos eliminar una máscara, debemos referenciar la propiedad **mask** a **null**: **objetoAEnmascarar.mask = null;**.

La función **cargarXML()** es la que se ocupa de cargar la **estructura XML**:

```
public function cargarXML():void {
     requestXML = new URLRequest();
     requestXML.url=”data.xml”;
     loaderXML=new URLLoader();
     loaderXML.load(requestXML);
     loaderXML.addEventListener(Event.COMPLETE, xmlCargado, false, 0,
true);
}
```

Las clases **URLRequest** y **URLLoader** ya las hemos visto en los capítulos anteriores. Aquí el funcionamiento es exactamente el mismo: definimos una petición y luego cargamos los contenidos. El nuevo concepto que tenemos aquí es la incorporación del Event Listener **Event.COMPLETE** para la clase **URLLoader**, a fin de detectar cuándo se completó la carga del **XML**. Al detectarse el evento, llamamos a la función **xmlCargado()**:

```
private function xmlCargado(event:Event):void {
     miXML = new XML(loaderXML.data);
     totalPics=miXML.*.length();
     picsArray = new Array();

     for (var m:uint = 0; m<totalPics; m++) {
            var ruta:String=miXML.child(m).@ruta;
            var desc:String=miXML.child(m).@desc;
            var picObject:Object={ruta:ruta,desc:desc};
            picsArray.push(picObject);
     }
     crearThumbs(0);
}
```

PROPIEDAD .LENGTH DE ARRAY Y MÉTODO .LENGTH() DE XML

Al utilizar **arrays** accedemos a la cantidad total de sus contenidos por medio de la **propiedad length**, mientras que para hacerlo en una estructura **XML** no se emplea una propiedad, sino el **método .length()**. Es importante conocer esta diferencia a fin de evitar confusiones a la hora de desarrollar y no mezclar la terminología de **arrays** con **XML**.

Lo primero que hacemos dentro de la función **xmlCargado()** es crear la instancia de la clase **XML** y le asignamos los contenidos (propiedad **data** de la clase **URLLoader**) que fueron cargados, como vemos a continuación:

```
miXML = new XML(loaderXML.data);
```

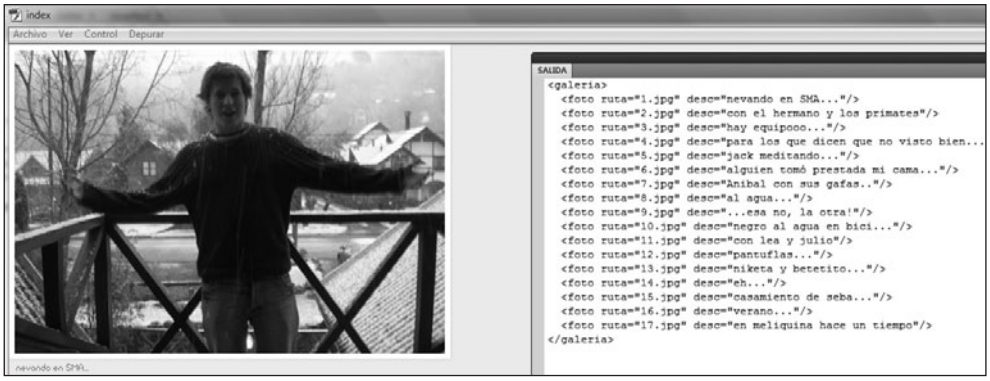


Figura 6. *Utilizar la función trace() puede ser de gran utilidad para verificar que los contenidos se estén cargando correctamente o verificar errores.*

Luego, definimos la variable **totalPics** y le asignamos el valor numérico del total de nodos que contiene nuestra estructura, es decir, la cantidad de imágenes que tendrá nuestra galería. Veamos cómo:

```
totalPics=miXML.*.length();
```

Dentro de la función **cargarXML()**, ésta es la parte fundamental de nuestro código:

```
picsArray = new Array();

for (var m:uint = 0; m<totalPics; m++) {
     var rutaVar:String=miXML.child(m).@ruta;
     var descVar:String=miXML.child(m).@desc;
     var picObject:Object={ruta:rutaVar,desc:descVar};
     picsArray.push(picObject);
}
```

Veremos que hay varios conceptos nuevos en esta pequeña función: en primer lugar, encontramos las **matrices** o **arrays**. Un **array** es un tipo de **variable** que

permite tener más de un valor en su interior, a los cuales podemos acceder por medio de un **índice**. Lo creamos por medio del constructor **new Array()** y usaremos solamente uno de sus métodos, el método **push()**, por medio del cual agregamos contenidos al **array**. Dentro de un array podemos almacenar cualquier tipo de dato: cadenas de texto, números, objetos, etcétera.
Hay una cuestión de importancia que debemos considerar. Generalmente, cuando necesitamos varios contenidos dentro de una posición de un array, se suelen emplear **arrays asociativos**. Un array asociativo es aquel que presenta una estructura donde los elementos poseen nombres a diferencia de índices numéricos. Por ejemplo:

```
var miArray:Array = new Array({nombre:"Make", edad:24});
```

Si bien anteriormente se solía desarrollar de este modo, no es recomendable su uso. Para crear estas estructuras con propiedades y valores, se deben emplear **objetos**:

```
var miObjeto:Object={nombre:"make",edad:24};.
```

Y luego sí, una vez que creamos un objeto y le asignamos las propiedades que queremos, lo añadimos al final de nuestro array por medio del método **push()**.

```
picsArray.push(picObject);
```

Por otro lado, utilizamos nuestro primer **bucle for**: una sentencia mediante la cual se define un **valor de inicio**, luego **una condición** y finalmente, en caso de cumplirse la condición, se determina cómo **continuamos** el ciclo. Para verlo de un modo más claro con nuestro ejemplo, el valor de inicio es el **0**:

```
for (var m:uint = 0
```

CONCEPTOS BÁSICOS: ARRAYS, BUCLE FOR, CONDICIONALES

Las explicaciones respecto a los conceptos que hacen a la programación (manejo de arrays, bucles, condicionales) son básicas. Damos por hecho que quien se acerque a un libro de este tipo conoce acerca de estos términos. En caso contrario, probablemente resulte conveniente interiorizarse en la sintaxis básica del lenguaje y luego continuar con la lectura.

Si lo que queremos es que se recorra la totalidad de los nodos del XML, la condición es que nuestra variable sea menor a **totalPics**, que es la cantidad total de nodos:

```
for (var m:uint = 0; m<totalPics
```

En caso de cumplirse la condición, incrementa la variable en uno (**m++**) y el código se volverá a ejecutar mientras se cumpla la condición:

```
for (var m:uint = 0; m<totalPics; m++) {
```

Por último, hacemos uso de los **objetos** que hemos mencionado anteriormente. Un objeto nos permite asignarle propiedades con sus respectivos valores:

```
var rutaVar:String=miXML.child(m).@ruta;
var descVar:String=miXML.child(m).@desc;
var picObject:Object={ruta:rutaVar,desc:descVar};
```

Una vez que tenemos definidos los pares propiedad/valor, debemos agregar cada objeto a nuestro array como ya hemos visto. Es importante que recordemos que cada uno de ellos contendrá la información necesaria para crear nuestra galería: la ruta de la imagen y su descripción.

```
picsArray.push(picObject);
```

Al haber creado una instancia de **XML** (**miXML**), ésta hereda todos sus **métodos** y **propiedades**. Por medio del método **child()** accedemos al nodo que especifiquemos. Al indicar la variable **m** (variable auto incrementable al producirse el bucle **for**), ésta será distinta por cada vuelta que realice el bucle, accediendo de este modo a todos los nodos de nuestra estructura. Por último, si revisamos la estructura XML, veremos que por cada nodo definimos dos atributos:

```
<foto ruta=”9.jpg” desc=”...esa no, la otra!”/>
```

Para acceder a esta información desde Flash, empleamos el identificador **@**, nuevo en ActionScript 3.0, que utilizamos para distinguir los atributos de un objeto XML indicándole sus nombres (**ruta** y **desc**, en nuestro caso):

```
var rutaVar:String=miXML.child(m).@ruta;
var descVar:String=miXML.child(m).@desc;
```

Una vez finalizada la carga y almacenados los objetos con sus propiedades dentro del array **picsArray**, nuestra próxima tarea será cargar los **thumbnails** de nuestra película.

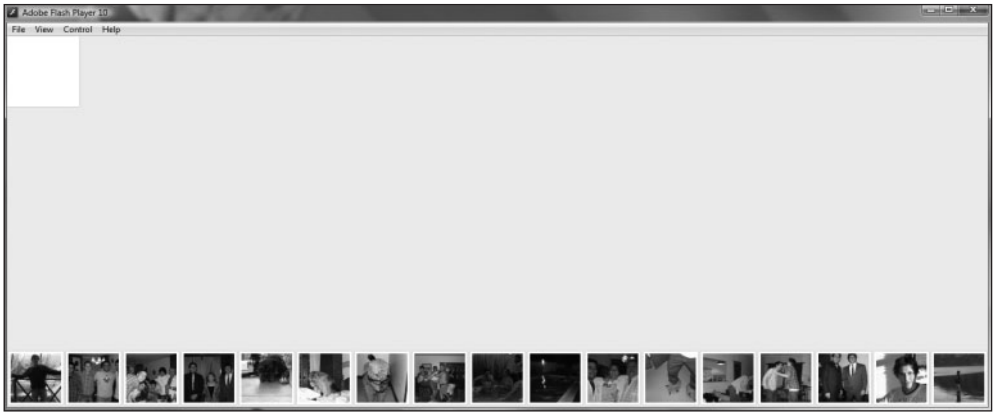


Figura 7. *Para cargar los* ***thumbnails*** *de nuestra galería tomaremos nuestras imágenes de la carpeta imagenes/thumbs. Las dimensiones de todas las miniaturas son de* ***75 px x 75 px.***

A partir de aquí, en lo que resta de nuestro código, nos ocuparemos puntualmente de lo que respecta al manejo de la imagen en Flash, ya que la mayor parte de la sintaxis empleada para el resto de los contenidos la hemos visto en los capítulos anteriores. Desde la función **crearThumbs()** vamos creando, uno a uno, los **MovieClips** que serán nuestros **thumbnails**. Generamos el **background** para cada uno, le asignamos una instancia de la clase **Loader**, lo enmascaramos por si llegamos a tener imágenes que superen las dimensiones predefinidas para los **thumbs** y, luego, procedemos a hacer la carga de cada imagen. A su vez, cada clip contiene un campo de texto que utilizaremos de **preloader** para indicar el porcentaje que se ha cargado de cada imagen.

THUMBNAILS PARA NUESTRA GALERÍA

Si bien contamos con propiedades que nos permiten modificar las dimensiones de nuestros objetos de visualización (**width** y **height**) y otras que nos dan la posibilidad de variar la escala (**scaleX** y **scaleY**), no es recomendable hacer uso de éstas sobre las imágenes, las cuales se romperán y perderán legibilidad. Lo ideal es generar una imagen para el **thumbnail** y otra distinta para el tamaño normal.

La lógica de la función **crearThumbs()** es la siguiente: el valor numérico que le pasamos de parámetro a la función equivale a cada una de las posiciones que contiene nuestro **array**, que almacenó todos los objetos con la información de cada imagen. Veámoslo a continuación:

```
crearThumbs(0);

private function crearThumbs(id:uint):void {
     actualPic = id;
```

Notaremos que al finalizar la carga, siempre y cuando la posición de la imagen que cargamos sea menor a la cantidad total de imágenes, volvemos a llamar a la función **crearThumbs()**, incrementando el valor del parámetro a pasarle (**++actualPic**):

```
if (actualPic+1<totalPics) crearThumbs(++actualPic);
```

VARIABLE ACTUALPIC	POSICIÓN DENTRO DE PICSARRAY	CONTENIDO DEL OBJETO
0	picsArray[0]	picObject.ruta = 1.jpg
		picObject.desc = “nevando en SMA...”
1	picsArray[1]	picObject.ruta = 2.jpg
		picObject.desc = “con el hermano y los primates”
2	picsArray[2]	picObject.ruta =3.jpg
		picObject.desc = “hay equipooo...”
3	picsArray[3]	picObject.ruta = 4.jpg
		picObject.desc = “para los que dicen que no visto bien...”
...	...	...

***Tabla 1.** Por cada vez que la variable **actualPic** incremente su valor, ingresaremos a la siguiente posición dentro del array **picsArray** y obtendremos las propiedades **ruta** y **desc** de cada uno de los objetos que creamos anteriormente.*

A los **MovieClips** les podemos asignar nuestras propiedades:

```
picContainer.ruta=picsArray[actualPic].ruta;
picContainer.desc=picsArray[actualPic].desc;
```

Como dijimos, la variable **actualPic** equivale a un valor numérico que se irá incrementando cada vez que se ejecute la función **crearThumb()**, por lo que cada vez que generemos un nuevo clip, éste obtendrá sus valores de ruta y descripción de la

siguiente posición dentro del array **picsArray**. Una vez creado el resto de los contenidos, éste es el código básico que necesitamos para generar una carga:

```
pic = new Loader();
var urlReq:URLRequest = new URLRequest();
urlReq.url="imagenes/thumbs/"+picContainer.ruta;
pic.load(urlReq);
```

El resto del código que empleamos con el objeto **pic** es para asignarle propiedades como lo hacemos con cualquier objeto de visualización: asignar su posición **x** e **y**, enmascarar por medio de la propiedad **mask**, establecer en **false** la propiedad **mouseEnabled**, modificar su transparencia por medio de **alpha**, etcétera.

Controlar la carga de imágenes: clase LoaderInfo

Veremos que haciendo uso de la clase **LoaderInfo** y de sus eventos, podremos generar aplicaciones mucho más robustas y fiables que nos faciliten un control prácticamente total sobre la aplicación que vayamos a desarrollar.
Conocer esta clase y sus métodos y propiedades no sólo nos permitirá ofrecerle una mejor experiencia al usuario sino tener un control mucho más exacto sobre todo aquello que está sucediendo durante nuestra carga, esos detalles que no siempre se ven en nuestra interfaz.

Clase LoaderInfo

La clase **LoaderInfo** nos brinda información sobre la carga de un archivo (SWF, JPG, GIF o PNG). Esta clase nos facilita eventos para que podamos saber cuándo se inició una carga, cuándo se produce un error durante ella, podemos conocer el progreso de esa carga (los bytes totales y los cargados), detectar cuándo se completó exitosamente o cuándo se finalizó la descarga de un contenido.

EVENTO	DESCRIPCIÓN
complete	Lo empleamos para saber cuándo el archivo se cargó correctamente.
httpStatus	Nos informa cuando se realiza una petición sobre HTTP.
init	Nos informa cuando podemos acceder a las propiedades y métodos de un archivo SWF que ha sido cargado.
ioError	Se ejecuta en caso de que se produzca un error en una operación de carga.
open	Por medio de este evento podemos saber cuándo se inicia una carga.
Progress	Lo utilizamos para conocer el estado de una carga; los bytes cargados y los bytes totales.
Unload	Utilizamos este evento cuando queremos eliminar un objeto que ha sido cargado anteriormente por medio del método load().

***Tabla 2.** Eventos de la clase LoaderInfo.*

Propiedad ContentLoaderInfo

La propiedad **contentLoaderInfo** de la clase **Loader** nos proporciona un objeto llamado **LoaderInfo** que contiene la información del archivo que se está cargando. Veamos nuestro código para saber cómo utilizar esta propiedad. Solamente haremos uso de tres listeners: uno para detectar cuando se completó la carga, otro para controlar el progreso de carga y otro para detectar algún error.

```
pic.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete, false, 0,
true);
pic.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR,
onErrorHandler, false, 0, true);
pic.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS, onProgress,
false, 0, true);
```

Evento IOErrorEvent.IO_ERROR

El evento **IOErrorEvent.IO_ERROR** lo empleamos para saber si se está intentando cargar un archivo que no existe. Más allá de que no informemos respecto al error de la carga al usuario, emplear este evento evitará que aparezca la ventana con el error en tiempo de ejecución de Flash.

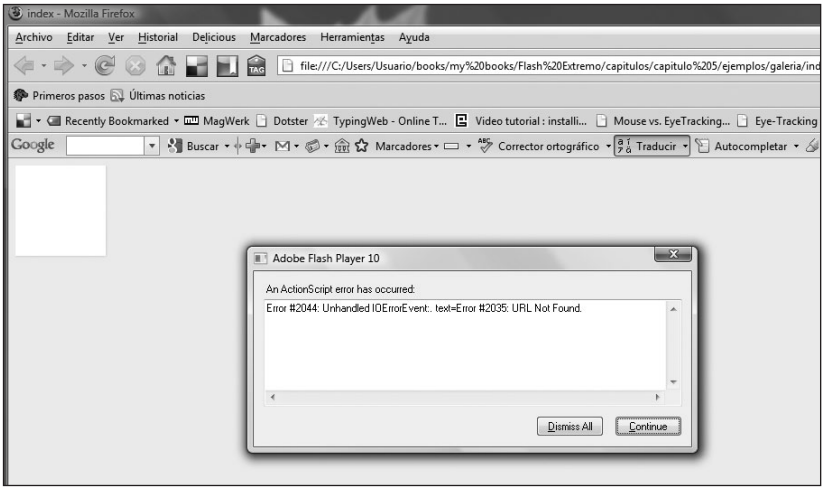


Figura 8. *Flash nos reporta un error al intentar cargar un archivo inexistente. Por medio del evento* ***IOErrorEvent.IO_ERROR****, podemos evitar este mensaje.*

Empleamos la función **onErrorHandler()** para que, en caso de que la imagen no se haya encontrado, se cree un campo de texto en tiempo de ejecución e indiquemos que esa imagen no se ha podido cargar. A su vez, eliminamos el evento de mouse para que no se pueda hacer clic en el objeto.

Figura 9. *En caso de no hallarse un archivo, establecemos que no se pueda hacer clic sobre él y generamos un campo de texto en tiempo de ejecución para informar al usuario al respecto.*

ProgressEvent.PROGRESS

Por medio de la detección de este evento podemos tener acceso a la información de bytes totales y bytes cargados de la imagen:

```
function onProgress(event:ProgressEvent):void {
     var cargado:Number=event.bytesLoaded;
     var total:Number=event.bytesTotal;
     var porcentaje:Number = Math.round((cargado/total) * 100);

     event.target.loader.parent.getChildByName("preloader").text=porcentaje.
     toString() + "%";
}
```

Evaluando esta información y estableciendo una sencilla regla de tres simple, podemos generar un **preloader** para informarle al usuario qué porcentaje de la imagen se cargó.

Figura 10. *Cada una de nuestras imágenes contendrá un preloader donde informamos del porcentaje de la carga por medio de un campo de texto.*

Event.COMPLETE

Por medio de este evento, detectamos cuando se completó la carga de un contenido.

```
function onComplete(event:Event):void {
     picContainer.removeChild(preloaderTxt);
     TweenLite.to(event.target.loader, 0.5, {alpha:1, overwrite:false});
if (actualPic+1<totalPics) crearThumbs(++actualPic);
}
```

Cuando la carga se ha completado, eliminamos el campo de texto que utilizamos de preloader y en caso de que el próximo número de imagen a subir sea menor al total de las imágenes, volvemos a llamar a la función **crearThumb()** incrementando en uno el valor de la variable **actualPic**.
Por último, dentro de la función **loadPic()** se encuentra el código que se ejecuta al hacer clic en cada uno de los **thumbnails**. El código necesario para generar una carga por medio del método **load()** ya lo hemos visto en reiteradas ocasiones. Es exactamente igual que el que necesitamos para generar la carga de las imágenes que se van a mostrar. La diferencia respecto al código anterior en los que cada imagen tiene su propio loader es que en esta fracción del código utilizamos un único loader para todas las imágenes, por lo que al cargar una nueva imagen debemos descargar la que había sido cargada anteriormente. Esto lo hacemos por medio del método **unload()**, como vemos a continuación:

```
mainPicLoader.unload();
```

En caso de no utilizar este método, el ejemplo seguirá funcionando, pero se cargará una imagen encima de la otra de forma constante. En cambio, empleando este método nos aseguramos de que se elimine la imagen anterior antes de mostrar la siguiente. Como hemos visto anteriormente, contamos con un listener para detectar cuando se completó la descarga:

```
mainPicLoader.contentLoaderInfo.addEventListener(Event.UNLOAD, load, false,
0, true);
```

Al completarse la descarga, llamamos a la función **load**:

```
function load(event:Event):void {
     var req:URLRequest = new URLRequest();
     req.url=”imagenes/normal/”+ ruta;
     mainPicLoader.contentLoaderInfo.addEventListener(Event.COMPLETE,
     onComplete, false, 0, true);
     mainPicLoader.load(req);
     isPic=true;
}
```

Lo importante de esta función es el Event Listener para detectar la carga. Al cargarse la imagen, llamamos a la función **onComplete**. En la función **onComplete** se

encuentra el código que utilizamos para modificar el tamaño del marco en base a las dimensiones de cada foto. Para conocerlas, accedemos por medio de las propiedades **width** y **height** del objeto **loader**:

```
rectWidth=event.target.loader.width;
rectHeight=event.target.loader.height;
```

Es importante saber que las dimensiones de una imagen (**width** y **height**) solamente están disponibles una vez que se completó la carga y no antes. Es por esto que necesitamos del evento **Event.COMPLETE**, ya que en caso contrario no sabríamos cuándo se completó la carga y no podríamos tener acceso a esta información.

Figura 11. *Transición: por medio de las propiedades width y height accedemos a las* ***dimensiones*** *en píxeles de la imagen cargada. Conociendo esa información, adaptamos el marco al tamaño de la foto.*

Si bien es cierto que podríamos seguir optimizando la galería aún más, lo importante es saber que con nuestro simple código podemos generar una galería que carga contenidos externos y que pesa menos de 20 kb. Pese a su sencillez, controlamos mediante la detección de eventos prácticamente todos los estados, desde la carga y su progreso hasta la detección de posibles errores. A continuación, iremos aún un poco más lejos e implementaremos la API de Flickr con un proyecto en Flash.

BÚSQUEDA BASADA EN LOS SERVICIOS DE FLICKR

En esta segunda parte veremos de qué manera podemos extender las posibilidades que nos brinda la Web para aplicarlas a nuestros desarrollos: generaremos un buscador de imágenes que tomará fotografías de **Flickr** (**www.flickr.com**), uno de los portales de mayor importancia en lo que a imágenes respecta.

Introducción a APIs

Desde que se comenzó a utilizar el concepto **Web 2.0** a fin de redefinir el nuevo destino que estaba tomando la Web, se dio origen a un diferente modo de pensar y de llevar a cabo los desarrollos. Las principales **webs 2.0** no sólo exponen sus contenidos en sus propios sitios, sino que, generalmente, ponen a nuestra disposición sus **APIs**. Básicamente, una **API** (**interfaz de programación de aplicaciones**) es un **servicio** creado a fin de que los desarrolladores podamos hacer uso de sus contenidos de forma relativamente sencilla. Por medio de estas APIs podemos acceder a ellos y manipularlos para desarrollar nuestras propias aplicaciones. **Flickr**, al igual que **Picasa**, **YouTube**, **Facebook**, **myspace**, **last.fm** y muchos portales más, cuenta con sus propias APIs. En este capítulo, emplearemos la API de Flickr para generar un buscador de imágenes.

Principales ventajas del uso de APIs

La principal ventaja de trabajar con las APIs es que hacemos uso de los recursos de la compañía prestadora del servicio; los contenidos los levantamos desde sus propios servidores, lo que implica que el espacio que estos ocupan, al igual que la transferencia necesaria para descargarlos, corren por parte de los servidores de quien pone a nuestra disposición sus APIs.

Otro de los grandes beneficios es la **estabilidad** de los servicios. Es de esperar que el mantenimiento y el correcto funcionamiento sean notables en vistas de que la mayoría de las compañías que poseen estos servicios cuentan con una cantidad de recursos considerables para que el funcionamiento sea el correcto (sabemos que **Picasa** y **YouTube** son de **Google** y que **Flickr** es de **Yahoo**, entre otros). Gracias a las APIs contamos con prácticamente toda la información que poseen estos servicios: millones de imágenes, millones de videos, etcétera.

Desventaja de las APIs

Su mayor **desventaja** es producto de su mayor ventaja: los contenidos se encuentran almacenados en los servidores de quien nos presta el servicio, por lo que el correcto funcionamiento de nuestros desarrollos depende del buen funcionamiento de los servidores y del servicio de quien nos brinda la API.

WWW.PICNIK.COM

El sitio web de **picnik** (**www.picnik.com**) es un gran ejemplo de lo lejos que se puede llegar implementando APIs desde Flash para editar, corregir, recortar, aplicar efectos sobre imágenes, etcétera. En el sitio no sólo se hace uso de las **APIs** de **Flickr**; también se utilizan las de **Picasa**, las de **photobucket**, las de **facebook** y las de **myspace**.

Conclusión

Es necesario conocer cuáles son las ventajas y las desventajas del uso de APIs para que, análisis mediante, sepamos cuándo es conveniente hacer uso de ellas y cuándo conviene optar por desarrollar la totalidad de los contenidos por nuestra parte. En algunas ocasiones, esta última opción sólo implica una mayor inversión de tiempo y una mayor cantidad de conocimientos, pero en otras hay varias limitaciones, tanto técnicas como económicas. En estos casos, debemos considerar el uso de APIs.

API de Flickr

Tratándose de un capítulo de imágenes, abarcaremos solamente la API de **Flickr** y utilizaremos sus servicios para generar un buscador de imágenes. En primer lugar, para hacer uso del servicio, debemos ingresar a la dirección **www.flickr.com/services/api/** y solicitar desde allí una **clave API**. En la parte superior del sitio se encuentra la opción para hacerlo.

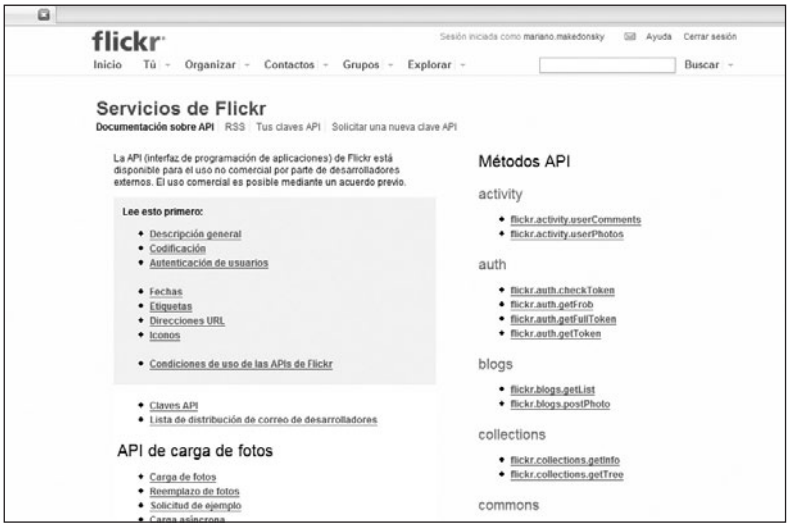


Figura 12. *Desde el sitio* ***www.flickr.com/services/api/*** *debemos solicitar la clave API para utilizar los servicios.*

API DE PICASA

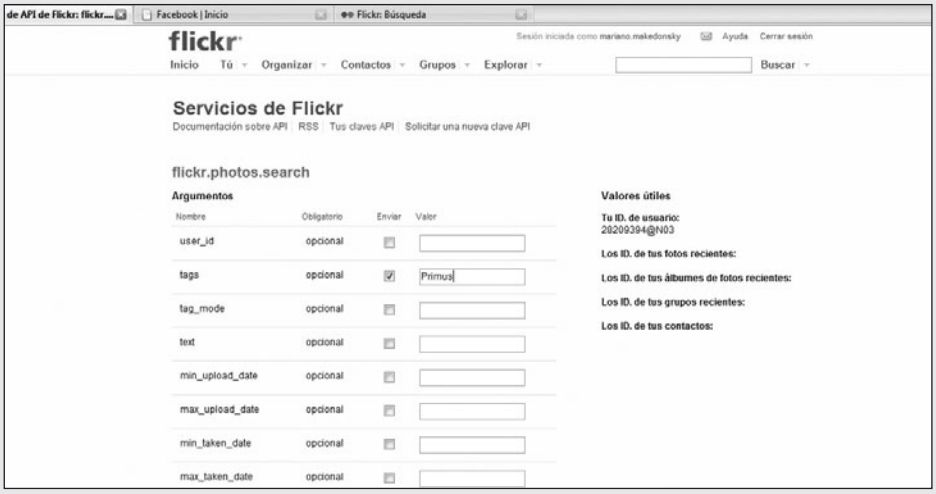
Si bien utilizaremos la API de Flickr, **Picasa**, la web de imágenes de **Google**, cuenta también con su API para que podamos usar sus servicios. Si estamos interesados en trabajar con ella, podremos encontrar toda la información necesaria para hacerlo en **http://code.google.com/intl/es/apis/picasaweb/developers_guide_protocol.html**.

Una vez que estamos dentro del sitio, debemos seguir tres sencillos pasos, completar nuestros datos y esperar la clave (**API key**) que **Flickr** nos facilitará, que es la que emplearemos siempre que queramos hacer algún desarrollo utilizando sus servicios. En **www.flickr.com/services/api/**, sobre el sector derecho, aparece una lista de opciones llamadas **Métodos API**. Estos son todos los recursos con los que contamos. Sería imposible abarcarlos en su totalidad, por lo que sólo veremos cómo implementar la opción **flickr.photos.search**, con la que generaremos nuestro buscador. Si hacemos clic sobre esa opción, seremos redireccionados a **www.flickr.com/services/api/flickr.photos.search.html** donde encontraremos toda la información relativa al uso de este servicio. En la parte inferior del sitio hay un link que indica **Explorador de API** y al hacer clic en él, iremos a **www.flickr.com/services/api/explore/?method=flickr.photos.search**, desde donde podemos generar nuestras pruebas con la API. Veamos cómo hacerlo.

■ Probar API de Flickr

PASO A PASO

1 Sobre la fila que lleva por nombre **tags**, indique la palabra que se va a buscar dentro del campo **valor** y active el **checkbox** que corresponde.



AS3FLICKRLIB

En el sitio **http://code.google.com/p/as3flickrlib/** encontraremos un proyecto desarrollado en **ActionScript 3.0** que nos proporciona una API para interactuar con todos los servicios de Flickr. No haremos uso de ella ya que sólo analizaremos los servicios de la Web. Sin embargo, puede ser de interés conocer en profundidad el funcionamiento de la librería **as3flickrlib**.

2 Diríjase a la parte inferior del sitio y presione el botón **Call Method...**. Verá que se realiza una búsqueda y se generán los resultados.

3 Debajo de esos resultados aparecerá una URL. Ábrala en una nueva ventana de su explorador para ver el XML resultante.

Lo que obtenemos es una **estructura XML** que contiene los resultados de la búsqueda que hemos realizado. En la primera parte del capítulo ya hemos visto de qué manera cargar estas estructuras. Ahora veamos de qué forma hacerlo para obtener un **XML** en base a la búsqueda que realicemos.

Integración de Flash con Flickr

Los conceptos referentes al tratamiento de imágenes y la carga de documentos XML los hemos visto en la primera parte del capítulo. A continuación aprenderemos de qué manera utilizar la API y la nueva sintaxis con la que cuenta el ejemplo.

Figura 13. *El resultado final de nuestra aplicación, utilizando los servicios de* ***Flickr*** *para generar un buscador de imágenes experimental.*

Cuando nuestras aplicaciones cargan contenidos externos situados en servidores distintos al que aloja a nuestra aplicación, debemos indicárselo a Flash para que sepa de qué dominios permitirnos la carga de contenidos. Esto lo hacemos por medio del método **allowDomain()**, de la clase **Security**:

```
Security.allowDomain(["api.flickr.com", "flickr.com", "*"]);
```

Dentro de la función **crearBusqueda()**, generamos el campo de texto para nuestra búsqueda, su respectivo botón y un campo de texto que nos informe respecto a los procesos que se estén llevando a cabo.

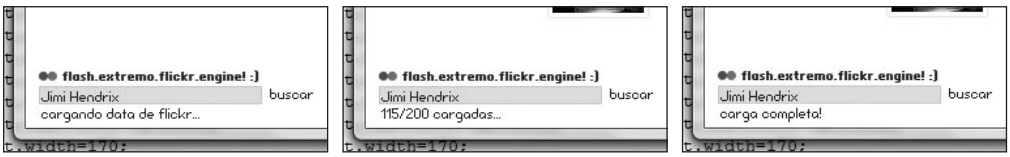


Figura 14. *Nuestro buscador de imágenes y sus distintos estados durante el proceso de búsqueda.*

Al presionar el botón que genera la búsqueda, llamamos a la función **checkVars()**, la cual por medio de una expresión regular controla que se haya ingresado algún carácter para la búsqueda. De ser así, llamamos a la función **loadFlickrData()**.

```
public function loadFlickrData():void {

     url="http://api.flickr.com/services/rest/?method=flickr.photos.
search&api_key="+apiKey+"&tags="+busqueda+"&per_page="+totalPics.
toString()+"&page="+pagina.toString();
     requestXML = new URLRequest();
     requestXML.url=url;
     loaderXML=new URLLoader();
     loaderXML.load(requestXML);
     loaderXML.addEventListener(Event.COMPLETE, xmlCargado, false, 0,
true);
}
```

Si bien no hay nada nuevo en esta función, lo que realmente nos interesa y es el eje central de nuestra aplicación es la **URL** a la cual estamos llamando:

```
url="http://api.flickr.com/services/rest/?method=flickr.photos.search&api_ke
y="+apiKey+"&tags="+busqueda+"&per_page="+totalPics.toString()+"&page="+pag
ina.toString();
```

Ésta es la cuestión central de nuestro código y la más importante. Al utilizar esta API, debemos definir una serie de variables al generar la llamada de la estructura XML. En primer lugar, la URL de los servicios que vamos a utilizar:

```
url="http://api.flickr.com/services/rest/"
```

Luego, en la variable **method**, debemos definir cuál de todos los servicios de la API vamos a emplear. Recordemos que para hacer una búsqueda, tenemos que utilizar el método **flickr.photos.search**:

```
url="http://api.flickr.com/services/rest/?method=flickr.photos.search
```

Posteriormente, concatenamos en la URL nuestra **API key** que solicitamos antes:

```
url="http://api.flickr.com/services/rest/?method=flickr.photos.search&api_ke
y="+apiKey
```

A continuación, debemos agregar a la llamada la variable **tag**, que contiene la etiqueta con la que queremos generar la búsqueda. A ella debemos concatenarle el valor de la variable **busqueda**, que será igual al del texto que se haya ingresado en el campo de texto **buscarTxt**, como vemos debajo:

```
url="http://api.flickr.com/services/rest/?method=flickr.photos.search&api_ke
y="+apiKey+"&tags="+busqueda
```

Por último, agregamos dos variables más. Una denominada **per_page**, por medio de la cual indicamos la cantidad de resultados que queremos que contenga nuestra consulta, y otra llamada **page**, que informa el número de página que vamos a cargar:

```
url="http://api.flickr.com/services/rest/?method=flickr.photos.search&api_ke
y="+apiKey+"&tags="+busqueda+"&per_page="+totalPics.toString()+"&page="+pag
ina.toString();
```

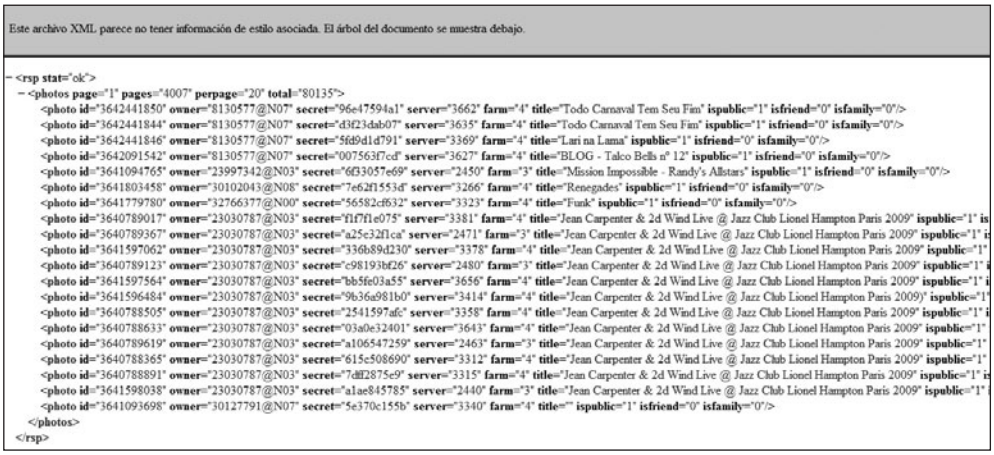


Figura 15. *Una vez creada la variable url, hacemos un trace() de ella, copiamos y pegamos el resultado en el navegador. Veremos la estructura XML que estamos cargando desde Flash.*

Una vez que la carga se completó exitosamente, contamos con la estructura XML dentro de nuestra película. Es importante saber que la información que esta estructura nos proporciona es todo lo que necesitamos para averiguar la ruta de cada una de las imágenes que vamos a cargar:

```
<photo id="3641779780" owner="32766377@N00" secret="56582cf632" serv
er="3323" farm="4" title="Funk" ispublic="1" isfriend="0" isfamily="0"/>
```

Respecto a la función **xmlCargado()**, ya conocemos su funcionamiento. Dentro de ella recorremos el XML y vamos almacenando objetos dentro de un array, los cuales contienen toda la información proporcionada en los atributos de cada nodo del XML. Asignamos los valores de los atributos a variables y luego las definimos como propiedades de un objeto llamado **picObject**. Se generará uno por cada nodo que haya en nuestra estructura y contendrá toda la información referente a cada imagen. Veamos el siguiente código:

```
var id:String=miXML.photos.child(m).@id;
var titulo:String=miXML.photos.child(m).@title;
var farm:String=miXML.photos.child(m).@farm;
var owner:String=miXML.photos.child(m).@owner;
var server:String=miXML.photos.child(m).@server;
var secret:String=miXML.photos.child(m).@secret;
var picObject:Object={id:id,farm:farm,titulo:titulo,owner:owner,secret:se
cret,server:server};
picsArray.push(picObject);
```

Una vez almacenados todos los objetos, llamamos a la función **cargarFotos()**. Si bien conocemos toda su sintaxis, hay una cuestión de importancia en lo que hace a la carga de la imagen. A cada MovieClip le asignamos una propiedad llamada **picPath** que, posteriormente, utilizaremos para cargar la imagen por medio del método **load()** de la clase **Loader**, como vemos a continuación:

```
picContainer.picPath =
"http://farm"+picsArray[m].farm+".static.flickr.com/"+picsArray[m].serv
er+"/"+picsArray[m].id+"_"+picsArray[m].secret;
```

Esta línea es, simplemente, la unión de varios de los valores que obtuvimos de los atributos del XML. Concatenando esta información (servidor en el cual se aloja la imagen, su id, etcétera), obtenemos la ruta final de la imagen. Veamos, en el siguiente ejemplo, la información que tomamos del XML:

http://**farm4**.static.flickr.com/**3627/3394184552_7246538b8c**.jpg

Otro dato que debemos tener en cuenta es que al momento de subir una imagen, Flickr hace una serie de **resampleos**, generando cinco tamaños distintos para una misma imagen. Para diferenciarlos, utiliza un carácter al final del nombre de la imagen a fin de identificarlas. En base a esta información, definimos en nuestro

código cuándo cargamos los cuadrados que **resampleó** Flickr de la imagen (**_s**) o cuándo las abrimos en su tamaño original (en nuestro caso, al detectarse el doble clic). Veamos la forma de nombrar cada tamaño de la imagen:

EXTENSIÓN	TIPO DE IMAGEN	EJEMPLO
_s	cuadrado	http://farm4.static.flickr.com/3347/3642629382_cc7d08f3f6_**s**.jpg
_t	thumbnail	http://farm4.static.flickr.com/3347/3642629382_cc7d08f3f6_**t**.jpg
_m	pequeña	http://farm4.static.flickr.com/3347/3642629382_cc7d08f3f6_**m**.jpg
(sin extensión)	mediana	http://farm4.static.flickr.com/3347/3642629382_cc7d08f3f6.jpg
_b	grande	http://farm4.static.flickr.com/3347/3642629382_cc7d08f3f6_**b**.jpg

***Tabla 3.** Las distintas extensiones que agrega Flickr al final de las imágenes, su significado y un ejemplo de cada una de ellas.*

En nuestro ejemplo, hacemos uso de los siguientes listeners a fin de detectar errores en la carga o si ésta se completó exitosamente:

```
pic.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete, false, 0,
true);
pic.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, onError,
false, 0, true);
```

A medida que las imágenes se van cargando, las colocamos en el centro del escenario para luego animarlas de manera aleatoria:

```
picContainer.x=(stage.stageWidth/2) - 32;
picContainer.y=(stage.stageHeight/2) - 32;
```

La función **getRandom()** que se encuentra al final de nuestro código se encarga de devolvernos un valor aleatorio estipulado entre un mínimo y un máximo que le indiquemos, como vemos a continuación:

```
private function getRandom(min:Number, max:Number):Number{
return Math.round((max - min) * Math.random() + min);
}
```

En nuestro ejemplo, utilizamos la función para que una vez concluida la carga, podamos definir los valores aleatorios **rotar** y **posicionar** sobre los ejes **x** e **y** de nuestras imágenes. A fin de mejorar el aspecto visual de nuestra galería, rotamos

las imágenes en 10 o en -10 grados y hacemos que se animen desde el centro hacia las coordenadas x e y obtenidas de manera aleatoria.

```
var rotate:Number = getRandom(-10, 10);
var posX:Number = getRandom(0, stage.stageWidth -
event.target.content.width);
var posY:Number = getRandom(0, 450 - event.target.content.height);

var bitmap:Bitmap = event.target.content;
bitmap.smoothing = true;
TweenLite.to(event.target.loader.parent, 0.5, {alpha:1, x:posX, y:posY, ro
tation:rotate});
```

***Figura 16.** Distintos valores aleatorios aplicados a la rotación y a las coordenadas **x** e **y** conforman una interfaz con una distribución y animaciones poco convencionales.*

Más novedades

Por último, algunos conceptos nuevos. Veremos que si bien los contenidos que se encuentran a continuación son de utilidad para nuestra galería, también nos pueden

SITIO DE EJEMPLO

El sitio japonés **www.fotologue.jp** está desarrollado íntegramente en **Flash**. Vale la pena visitarlo ya que es un gran ejemplo de los desarrollos que se pueden llevar a cabo haciendo un buen uso de **ActionScript**. A su vez, encontraremos muchos de los conceptos vistos en este capítulo en lo que a tratamiento de imágenes respecta.

ayudar en cualquier tipo de desarrollo y mejorar nuestras interfaces, tanto su aspecto visual como su funcionamiento, así como también proporcionarnos nuevas ideas para otros desarrollos que llevemos a cabo.

Aplicar smoothing a los mapas de bits

Ya hemos visto en el **capítulo 3** que por medio de la clase **Bitmap** podemos convertir a un objeto de nuestra lista de visualización en un mapa de bits. Lo que hacemos con estas líneas es transformar en un **bitmap** a la imagen que acabamos de cargar.

```
var bitmap:Bitmap = event.target.content;
bitmap.smoothing = true;
```

Realizamos esta acción porque la clase contiene una propiedad denominada **smoothing** que se encarga de aplicar un **suavizado** cuando generamos algún tipo de escala en nuestros objetos. Por haber rotado anteriormente el clip, aplicarle **smoothing** (suavizado) mejora considerablemente la calidad de la imagen. Un buen modo de notar las mejoras visuales que nos permite esta propiedad es comentando la línea de código que se encarga de aplicar el suavizado y, luego, probar nuestro ejemplo. Notaremos una reducción considerable en la calidad de las imágenes.

Definir índice para objetos (setChildIndex)

Por medio de los métodos **startDrag()** y **stopDrag()** hacemos que las imágenes sean **dragueables**. Aunque no hay nada nuevo hasta aquí, se nos presenta un problema: al cargar muchas imágenes y tener una cantidad considerable de contenidos en distinta profundidad, algunos se ubicarán por encima de otros. Por medio de la siguiente línea conseguimos que al hacerle clic a un objeto, éste se ubique en la última posición de la lista de visualización, quedando por encima del resto:

```
private function mouseDown(event:MouseEvent):void{
     var thisSprite:Sprite = Sprite(event.target);
     thisSprite.startDrag();
     mainContainer.setChildIndex(thisSprite, actualPic-1);
}
```

Doble clic en Flash

Antes, cuando queríamos hacer doble clic sobre un objeto, debíamos programar todos los comportamientos. ActionScript 3.0 nos facilitó por completo esta tarea, y a través de la propiedad **doubleClickEnabled** podemos definir, por medio de un valor booleano, si nuestro objeto permitirá que se haga **doble clic** sobre él o no:

```
picContainer.doubleClickEnabled = true;
```

Luego, asignar el listener para el doble clic es exactamente igual que para el resto de los eventos, como podemos ver a continuación:

```
picContainer.addEventListener(MouseEvent.DOUBLE_CLICK, doubleClick, false,
0, true);
```

De este modo, la función **doubleClick()** se ejecutará en respuesta a un doble clic sobre cualquiera de nuestros objetos.

El reemplazo para getURL(); la función navigateToURL();

Por último, y para concluir este ejemplo, hacemos que al detectarse el doble clic se abra en una ventana nueva del explorador la imagen en su tamaño normal:

```
private function doubleClick(event:MouseEvent):void{
     var myReq:URLRequest = new URLRequest();
     myReq.url = event.target.picPath + ".jpg"
     navigateToURL(myReq,"_blank");
}
```

En las versiones anteriores de Flash, para ir a una URL, lo hacíamos por medio de **getURL()**. Ahora, debemos emplear la función **navigateToURL()**, que acepta **dos parámetros**. En el primero de ellos indicamos una instancia de la clase **URLRequest**, que es la que contiene la URL a la cual queremos ir. El segundo parámetro (de carácter opcional) nos permite definir de qué manera se abrirá la ventana (**"_self"** para la misma ventana, **"_blank"** para una nueva ventana).

... RESUMEN

El manejo de imágenes puede ser tan sencillo o complejo como nosotros lo determinemos. Las condiciones están dadas tanto en la Web como en el lenguaje AS3 para llevar nuestros desarrollos a límites que un tiempo atrás eran difíciles de imaginar. Las APIs nos permiten extender nuestros proyectos y combinar nuestras capacidades con los millones de datos que hay en la Web. En el próximo capítulo abarcaremos el manejo de sonidos y espectros de sonido.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1) ¿Cuáles son las diferencias entre utilizar imágenes importadas e imágenes externas?

2) ¿Qué método nos permite optimizar el peso de los archivos y lograr una más sencilla actualización de los contenidos?

3) ¿De qué se encarga la clase **Loader**? ¿Cuál es la función de su método **load()**?

4) ¿Cuál es la ventaja de trabajar con estructuras XML? ¿Qué clase necesitamos para crear un nuevo objeto XML? ¿Para qué tipo de desarrollos es conveniente utilizarlas?

5) ¿Cómo se realiza la carga de un XML?

6) ¿Qué listener utilizamos para detectar el progreso de carga de una imagen o archivo .SWF externo?, ¿cuál para detectar un error en la carga y cuál para detectar el fin de la carga?

7) ¿Qué es la clase **LoaderInfo**? ¿Cuáles son sus eventos y para qué sirven? ¿Para qué se aplica la propiedad **contentLoaderInfo**?

8) ¿Qué evento usamos para detectar un error en la carga y cuál para conocer su progreso?

9) ¿Cómo accedemos a información del ancho (**width**) y del alto (**height**) de las imágenes? ¿Cuándo está disponible esta información?

EJERCICIOS PRÁCTICOS

1) Genere una galería de imágenes, cargando los contenidos de manera externa. Elija una distribución de los contenidos distinta a la que empleamos en nuestro ejemplo.

2) Aplique los Event Listeners necesarios para controlar los distintos estados del proceso de carga de una imagen: carga completa, progreso de carga y posibles errores.

3) Genere un preloader que no sea numérico para la imagen principal; aplique el porcentaje de la carga a la escala de algún Sprite o MovieClip o invente algún preloader no convencional.

4) Investigue respecto a la API de Picasa.

5) En base a los conocimientos adquiridos en la segunda parte del capítulo, intente generar un buscador de imágenes utilizando los servicios de Picasa.

Sonido en Flash y espectros de sonido

En este capítulo abordaremos los conceptos necesarios para comprender la implementación de audio en nuestras películas y cómo hacer un uso adecuado de él. También conoceremos la nueva sintaxis que ActionScript 3.0 nos propone para manejarlo, y desarrollaremos un reproductor de MP3. Para concluir, veremos cómo acceder a la información de las ondas de sonido para crear representaciones visuales del sonido.

SONIDO EN FLASH

Generalmente, se tiende a cometer dos errores en la implementación del audio: se lo relega por completo o se lo emplea del modo inadecuado. Bien utilizado, el sonido es un elemento esencial en el esquema de comunicación de la gran mayoría de los desarrollos. La intención de este capítulo es abordar los sonidos, conocer los procesos por medio de los que podemos optimizar su uso y entender las ventajas de cada caso. Finalmente, desarrollaremos un **reproductor** de MP3 y haremos uso del método **computeSpectrum()** de la clase **SoundMixer** incluida en ActionScript 3.0, por medio de la cual podemos obtener información sobre la amplitud del audio y, de esta manera, crear representaciones visuales del sonido.

Sonidos internos y externos

Al igual que sucede con las imágenes y los videos, el audio lo podemos tratar de manera interna o externa, siendo sus ventajas y desventajas en cada caso iguales a las que vimos en los capítulos anteriores al trabajar con imágenes y como veremos luego con los videos, tanto de manera interna como externa.

Sonidos internos

Del mismo modo que las imágenes, podemos importar sonidos a la biblioteca o al escenario. Para hacerlo, debemos ir a **Archivo**, seleccionar del menú la opción **Importar** y, luego, podemos elegir entre **Importar a escenario...** o **Importar a biblioteca...**.
Al importar sonidos al escenario, a diferencia de las imágenes que se posicionan sobre la escena, el audio se ubica en el primer fotograma de nuestra película. Si presionamos la tecla **F5** sobre la línea de tiempo, se irán agregando fotogramas y veremos la información del audio. De todos modos, es importante mencionar que si no colocamos un **stop()** al final de la línea, el cabezal reproducirá de nuevo el sonido cada vez que se posicione en el primer fotograma.

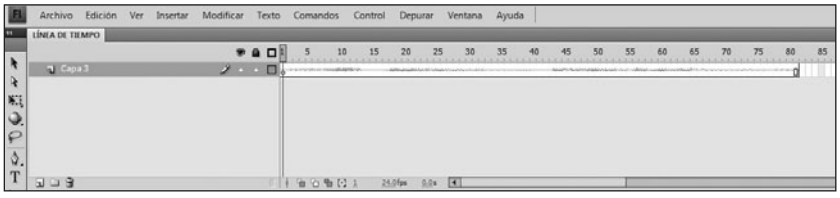


Figura 1. *Al importar sonidos al escenario, estos se colocarán sobre los frames.*

Si abrimos la biblioteca (**F12**) y presionamos el botón derecho sobre el MP3 que importamos, podemos acceder a **Propiedades...**. Desde ellas, si quitamos la tilde de la opción **Usar calidad de MP3 importado**, se nos desplegará un menú desde el cual podemos modificar la **Velocidad de bits del sonido** y su **Calidad**.

Figura 2. *Desde el panel* ***Propiedades de sonido*** *podemos modificar la* ***velocidad de los bits*** *y la* ***calidad*** *de los archivos de audio.*

Cuando desarrollamos nuestros proyectos, pocas veces nos va a ser de utilidad incluir el audio dentro de un fotograma; generalmente, es conveniente crear una instancia de dicho objeto en la biblioteca y manipularlo como lo hacemos con cualquier objeto. Antes de ver de qué manera hacerlo, debemos abarcar algunos conceptos nuevos en lo que se refiere al manejo de sonidos.

Sonidos externos

Los sonidos externos nos proporcionan una serie de ventajas sobre los sonidos internos. A continuación veremos sólo algunas de ellas, que evidencian de forma suficiente la conveniencia de utilizar este modo:

- El tamaño de nuestro SWF final no varía, independientemente de la cantidad de sonidos que carguemos en nuestra película.
- La actualización es sencilla; simplemente basta reemplazar el audio externo y Flash lo levantará cada vez que se ejecute.
- Podemos tomar decisiones en tiempo de ejecución: si el equipo que está corriendo nuestra película no cuenta con soporte para audio, ¿qué sentido tendría hacerle cargar al usuario los kb o Mb que pese nuestro archivo de sonido? Accediendo a esta información, podemos definir si es conveniente o no cargar el audio.
- Podemos calcular la velocidad de conexión del usuario y, en base a ella, definir el almacenamiento de datos en el buffer antes de reproducirlo.

Veamos, ahora, cómo cargar un sonido externo: creamos un objeto de la clase **Sound** al que llamaremos **sound**; luego por medio del método **load()** indicamos el archivo a cargar y, por medio del método **play()**, le damos comienzo a la reproducción.

```
var req:URLRequest = new URLRequest();
req.url = "mp3/sound.mp3";

var sound:Sound = new Sound();
sound.load(req);
sound.play();
```

Al método **play()** podemos indicarle tres parámetros adicionales: **play(comienzo, loops, soundStransform);**. En el primer parámetro, podemos definir el **milisegundo** en el que va a comenzar a reproducirse nuestro sonido. Si bien por defecto su valor será **0**, pronto veremos que necesitamos hacer uso de este parámetro al reanudar una reproducción. En el parámetro **loop**, podemos indicar la cantidad de veces que queremos que el sonido se reproduzca. Por último, el parámetro **soundTransform** nos permite establecer una instancia de la clase **SoundTransform** por medio de la cual podemos definir el **volumen** o el **paneo** del audio. No haremos uso de este parámetro ya que veremos que hay otra forma de utilizar esta clase y aplicar cambios de volumen y paneo.

CREACIÓN DE UN REPRODUCTOR DE MP3

El mejor modo de entender las nuevas clases que nos brinda ActionScript 3.0 para el manejo de audio, al igual que sus propiedades y métodos, es por medio de un ejemplo práctico. A continuación desarrollaremos un **reproductor de MP3** que contendrá:

- Stop y play para el sonido.
- Control de volumen.
- Control de streaming.
- Barra de tiempo transcurrido.
- Contador de tiempo total y tiempo transcurrido.
- Lectura de etiquetas ID3.
- Representaciones visuales del sonido.

En esta primera parte del capítulo abarcaremos los primeros seis de estos puntos, y en la segunda veremos el nuevo método **computeSpectrum()** y de qué manera crear representaciones visuales de la frecuencia o de la amplitud del sonido.

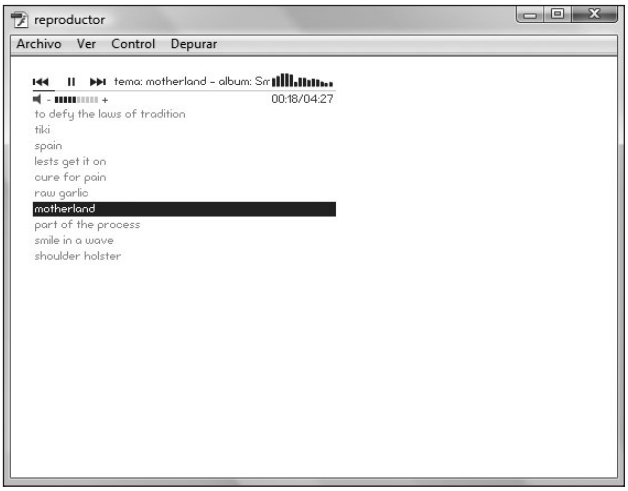


Figura 3. *Resultado final de nuestro desarrollo; a continuación veremos cómo creamos todas sus funciones.*

Los temas referentes a la carga y al tratamiento de estructuras XML ya los hemos visto en el capítulo anterior, por lo que aquí nos abocaremos, únicamente, a lo que implique manejo de sonidos y nueva sintaxis:

Clase **SoundMixer**
Clase **SoundLoaderContext**
Clase **SoundChannel**
Clase **SoundTransform**

Detener todos los sonidos

ActionScript 3.0 cuenta con una nueva clase denominada **SoundMixer**, que nos facilita propiedades y métodos para tener un control global de los sonidos que se están reproduciendo. Al tratarse de un reproductor, necesitaremos que al iniciar un nuevo tema, en caso de que se esté reproduciendo otro, se detengan todos los sonidos. Esto lo hacemos por medio del método **stopAll()**:

```
SoundMixer.stopAll();
```

Definir el tiempo de buffer: clase SoundLoaderContext

Por medio de la clase **SoundLoaderContext**, utilizándola en conjunto con el método **load()** de la clase **Sound**, podemos especificar la cantidad de segundos que deseamos que se almacenen en el buffer antes de comenzar a reproducir un sonido:

```
var context:SoundLoaderContext = new SoundLoaderContext(5000);
miSonido = new Sound(
miSonido.load(request, context);
```

A la clase debemos indicarle el número de milisegundos que deseamos almacenar antes de que comience la reproducción. Como sabemos, 1000 milisegundos equivalen a 1 segundo. Entonces, definiendo este valor en 5000, nos garantizamos que habrá 5 segundos de carga antes de que comience la reproducción del sonido. Quizás lo más interesante, más allá de establecer un tiempo de almacenamiento en buffer al azar, sería deducir la velocidad de descarga con la que cuenta el usuario y, en base a esa información, definir el tiempo necesario que conviene almacenar para que el usuario pueda reproducir un sonido sin interrupciones. Si bien ActionScript 3.0 no cuenta con documentación oficial para esto, hay varios tutoriales en Internet que brindan información al respecto y que nos serán de mucha utilidad para implementar esta funcionalidad.

¿Cómo comenzar la reproducción de un sonido?

La nueva clase **SoundChannel** se encarga de controlar un sonido de nuestra aplicación. Como su nombre lo indica, **SoundChannel** referencia a un **canal de sonido**, de manera tal que cada vez que queramos reproducir un nuevo sonido, deberemos asignarle un nuevo canal. Una de las principales ventajas que nos brinda tener separados los sonidos en distintos canales, es que nos facilita considerablemente su manipulación en caso de contar con más de un sonido en nuestra aplicación. Entonces, cada vez que deseamos comenzar una reproducción, debemos asignarle nuestro sonido (**miSonido**) a nuestro canal (**miCanal**). De este modo, comenzará la reproducción.

```
miSonido = new Sound()
miSonido.load(request, context);
miCanal = miSonido.play();
```

¿Cómo detener o reanudar la reproducción?

La clase **SoundChannel**, a su vez, nos provee del método **stop()**. Como su nombre lo indica, lo utilizamos para detener la reproducción del audio. En nuestra interfaz gráfica, contamos con un botón denominado **playIcon** al que le asignamos un Event Listener para detectar cuándo se está haciendo clic sobre él. Veamos, en la página siguiente, el código para lograrlo:

```
playIcon = new Play();
playIcon.addEventListener(MouseEvent.MOUSE_DOWN, stopAndPlay, false, 0,
true);
mainContainer.addChild(playIcon);
```

La función **stopAndPlay**, detendrá o reanudará el sonido según corresponda:

```
private function stopAndPlay(event:MouseEvent):void{
     var position:Number = miCanal.position;
     if(isPlaying){
            miCanal.stop();
            isPlaying = false;
            event.target.gotoAndStop(1);
     }else{
            event.target.gotoAndStop(3);
            miCanal = miSonido.play(position);
            isPlaying = true;
     }
}
```

Si bien sabemos que por medio del método **stop()** de la clase **SoundChannel** detenemos la reproducción de un sonido y que por medio del método **play()** de la clase **Sound** lo reanudamos, necesitamos conocer la **posición** en la que se encuentra la reproducción para que al reanudarla lo haga desde ese punto. En caso contrario, cada vez que reanudemos la reproducción, lo hará desde 0 y se reproducirá todo el tema nuevamente. Podemos conocer la posición en la que se encuentra nuestra reproducción por medio de la propiedad **position** de la clase **SoundChannel**, que nos indica el **punto actual** en el que se encuentra la reproducción de un sonido. En caso de que ésta haya sido detenida, la propiedad **position** indicará el **último punto** que se reprodujo de nuestro audio. Sabiendo esto, averiguamos la posición al presionar el botón de pausa:

```
var position:Number = miCanal.position;
```

Y como vimos anteriormente, al método **play()** de la clase **Sound** podemos indicarle la posición en la que queremos que comience la reproducción:

```
miCanal = miSonido.play(position);
```

De esta manera, le asignamos el valor de la variable **position** y el sonido reanuda desde su último punto. Por último, por medio del método **gotoAndStop()**, movemos el cabezal dentro del clip **playIcon**. Cuando el sonido se reproduce, le asignamos el signo de **pausa**, y cuando está en pausa, el signo **play** para que se reanude la reproducción.

¿Cómo detectar el final de la reproducción de un sonido?

La clase **SoundChannel** nos proporciona el evento **soundComplete**. Como su nombre lo indica, este evento se distribuye cuando finalizó la reproducción de un sonido:

```
miCanal.addEventListener(Event.SOUND_COMPLETE, sonidoCompleto, false, 0,
true);

private function sonidoCompleto(event:Event):void{
     nextSong(undefined);
}
```

¿Cómo asignar y modificar el volumen a un sonido?

Para asignar y modificar el volumen de un sonido, lo hacemos por medio de la propiedad **volume** de la clase **SoundTransform**, nueva en ActionScript 3.0. La propiedad **volume** acepta valores entre el **0** y el **1**, donde **0** es el silencio y **1** el máximo volumen. Próximamente haremos uso de una clase que nos permitirá asignar, por medio de un **slider**, los valores intermedios **0.1**, **0.2**, **0.3**, **0.4**, etcétera, a fin de modificar el sonido de nuestra aplicación.

```
miVolumen = new SoundTransform();
miVolumen.volume = 1;
```

Una vez que definimos nuestra instancia, debemos aplicarla a nuestro canal. Para hacerlo, empleamos la propiedad **soundTransform**:

```
miCanal.soundTransform = miVolumen;
```

Paneo de sonidos

Por medio de la propiedad **setPan** de la clase **SoundTransform**, es posible modificar el desplazamiento lateral del sonido (**paneo**), con valores de **-1** para el lateral

izquierdo y de **1** para el lateral derecho. Por defecto, su valor es **0**, lo que indica que no hay ningún tipo de desplazamiento. No haremos uso de esta opción en nuestra aplicación, considerando que, al tratarse de un reproductor de MP3, es de esperar que los paneos de audio se hayan hecho en el momento en el que el autor del tema editó la pista de audio. Pero, en algunos casos particulares, podríamos necesitar hacerlo. Pensemos en el siguiente ejemplo: una pelotita se encuentra picando sobre el escenario y al producirse el pique, reproducimos un sonido. Manejando correctamente el paneo del audio y con un condicional para detectar dónde se produjo el pique, podemos hacer que el sonido salga por el parlante izquierdo cuando el pique sea de la mitad del escenario hacia la izquierda o que salga por el parlante derecho, cuando el pique sea de la mitad hacia la derecha. O incluso, con algunos conocimientos de matemática, podemos mejorar aun más esta experiencia, equiparando el largo de esta aplicación con valores del -1 al 1, de manera tal que el paneo del audio equivalga a la posición en la que pique la pelotita. A modo de ejemplo, en caso de que el pique sea en el centro del escenario, el valor de paneo será **0**, o en caso de que el pique sea en 3/4 de nuestra aplicación, el paneo será **0.75**, aproximadamente.

Mejorar la experiencia del sonido

Si bien el **0** representa el silencio en el audio y el **1** es el volumen máximo, estos valores nos serían útiles sólo si implementamos un botón para silenciar (**mute**) nuestra aplicación. Podemos enriquecer nuestro desarrollo generando un control de sonido que le permita al usuario una mayor interacción con nuestra aplicación y brindarle la posibilidad de **silenciar** el reproductor y **subir** o **bajar** el volumen, ya sea por medio de los botones correspondientes o bien desplazando la barra de sonido. La clase **Volume** que se encarga de generar el control del volumen se encuentra dentro del paquete **com.utils** (recordemos que podemos descargar este material desde el sitio de la editorial en **www.redusers.com** o desde **http://as3hispano.com**).

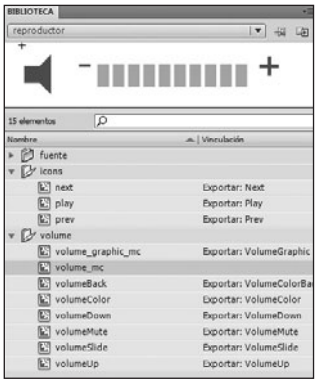


***Figura 4.** Por medio del control de sonido es posible modificar el volumen entre el mínimo y el máximo, o bien silenciar la aplicación.*

Hablamos en reiteradas ocasiones de las ventajas de emplear clases en nuestros desarrollos, y éste es un buen ejemplo para justificar el porqué. En verdad, un control de sonido es realmente útil y no sólo sirve para un reproductor de MP3; también podemos aplicarlo a un reproductor de video e incluso, haciendo unos pequeños cambios, podríamos convertir nuestro **slider de sonido** en un **slider genérico** que produzca una serie de valores entre un mínimo y un máximo que definamos. Como podemos imaginar, este concepto es aplicable para efectuar transformaciones en cualquier objeto, propiedad o método que acepte valores numéricos dentro de un rango.
Entonces, cuando tengamos que emplear un control de volumen, en lugar de desarrollarlo cada vez que un proyecto lo requiera o tener que cortar y pegar código, simplemente creamos una **instancia** de nuestra **clase** y hacemos uso de ella:

```
myVolume = new Volume(0x333333, 0xcccccc);
myVolume.setParams(0.5);
myVolume.addEventListener(Volume.VOLUME_SETTER, volumeSetter);
myVolume.x = 0;
myVolume.y = 15;
mainContainer.addChild(myVolume);
```

Estas pocas líneas de código que vimos son todo lo que necesitamos para crear una instancia de nuestra clase, pasarle un parámetro, posicionar nuestro control de volumen en el escenario y añadirlo por medio del método **addChild()** al contenedor de nuestra película. De no hacerlo de este modo, a nuestro código de la clase **Mp3Player** deberíamos agregarle las más de doscientas líneas que ocupa el código de la clase **Volume**. Si bien éste es un desarrollo pequeño, imaginemos hasta qué punto podríamos hacerlo complejo, agregándole una enorme cantidad de funcionalidades al reproductor. A medida que crezca, mayor será nuestra necesidad de distribuir correctamente los contenidos.
A su vez, otra ventaja de utilizar una clase es que una persona con conocimientos básicos puede hacer uso de un control de volumen sin necesidad de adentrarse en la dificultad del código ni de entender su funcionamiento. Simplemente, agregando la clase y conociendo sus funcionalidades, en pocas líneas se obtiene un control de volumen sin tener que programar demasiado.
No es nuestro interés analizar cómo hacer un control de sonido y que sus valores vayan del **0** al **1**; con conocimientos básicos de matemática, encontraremos las respuestas dentro de la clase **Volume**; sus funciones son realmente sencillas y no hay nada que no hayamos visto hasta el momento. Lo que nos interesa es ver una de las maneras por medio de las cuales podemos **comunicar dos clases**. O mejor dicho, de qué modo pasar parámetros de una clase a otra.

La clase **Volume** será la encargada de generar un valor numérico entre el **0** y **1**. Ahora bien, ese valor lo necesitamos dentro de la clase principal de nuestro desarrollo, **Mp3Player**, que es donde se encuentra nuestra instancia de la clase **SoundTransform** y donde tenemos que aplicar el volumen.

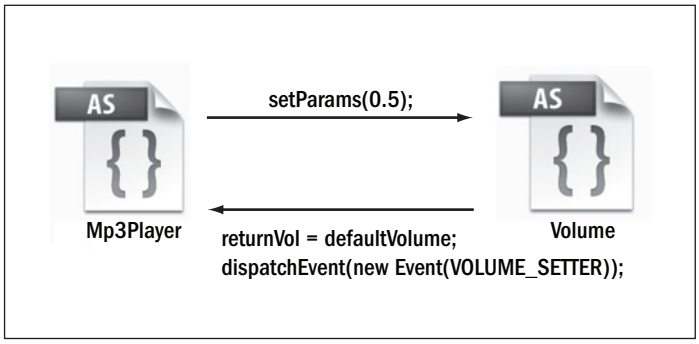


Figura 5. *Flujo de datos; comunicación entre la clase principal Mp3Player y la clase Volume.*

Enviar valores de una clase a su **subclase** es realmente sencillo: una vez definido su constructor, establecemos el nombre del método al que queremos llamar y le pasamos los parámetros que sean necesarios. Desde la clase **Mp3Player**, una vez creada la instancia (**myVolume**) de nuestra clase **Volume**, utilizamos su método **setParams()** para pasarle un parámetro. Esto en verdad es sencillo:

```
myVolume.setParams(0.5);
```

Para recibir el valor que estamos pasando desde la clase **Mp3Player**, necesitamos una función que acepte esa variable dentro de la clase **Volume**:

```
public function setParams(defaultVolume):void{
     this.defaultVolume = defaultVolume;

     sendVolume(defaultVolume);
}
```

Observemos que la función **setParams()** es declarada de **carácter público** (**public**). De este modo, definimos que se puede acceder a ella desde cualquier parte de nuestro código. Hasta ahora, hemos utilizado la palabra clave **public** solamente cuando definimos la clase y para su constructor (ambas son siempre públicas), y hemos definido el resto de las funciones como privadas.

PALABRA CLAVE DE ATRIBUTO	DESCRIPCIÓN
public	Utilizamos la palabra clave **public** cuando queremos establecer que una clase, una variable o un método estén disponibles desde cualquier parte de nuestro código. Un claro ejemplo es la clase **Volume** con la que estamos trabajando. Definimos su función **setParams()** como pública porque si no, no podríamos acceder a ella desde la clase **Mp3Player**.
internal	Cuando definimos una clase, una variable, una constante o una función con la palabra clave **internal,** hacemos que estén únicamente disponibles para cualquier llamada que se produzca dentro de un mismo paquete.
private	Cuando definimos una variable, constante o método por medio de la palabra clave **private**, estos estarán únicamente disponibles para la clase dentro de la que se declaran o definen. No podemos acceder desde otras clases; solamente es posible emplearlas dentro de la clase que las contiene.
protected	Utilizamos la palabra clave **protected** para especificar que una variable, una constante o un método están disponibles solamente para la clase que los contiene y para las subclases de esa misma clase.

Tabla 1. *Tabla con las distintas palabras clave para atributos y su descripción.*

No tiene ninguna complejidad pasar variables de una clase a una subclase. El problema, generalmente, se presenta cuando queremos pasar un valor de una subclase a una clase. En estos casos, debemos crear nuestros propios eventos. Para generar un evento, lo hacemos del mismo modo en el que aplicamos Event Listeners para cualquier circunstancia. Recordemos que al declarar la instancia de la clase **Volume** (**myVolume**), le asignamos un Event Listener:

```
myVolume.addEventListener(Volume.VOLUME_SETTER, volumeSetter, false, 0,
true);
```

La diferencia radica en que el tipo de evento lo creamos nosotros mismos, definiendo el nombre de la clase y el evento que se ejecutará:

```
Volume.VOLUME_SETTER, volumeSetter
```

De este modo, si bien hemos declarado el evento, debemos generar el código que se encargue de ejecutarlo. Para lograrlo, en primer lugar, tenemos que definir la constante dentro de la clase **Volume**:

```
public static const VOLUME_SETTER:String = “VOLUME_SETTER”;
```

Una vez definida, para ejecutarlo utilizamos el método **dispatchEvent()**, indicando como parámetro el evento que queremos que se ejecute:

```
dispatchEvent(new Event(VOLUME_SETTER));
```

De este modo, al producirse el **dispatchEvent** dentro de la subclase **Volume**, se estará llamando a la función **volumeSetter** de la clase principal, **Mp3Player**:

```
private function volumeSetter(event:Event):void{
     miVolumen.volume = myVolume.returnVol;
     miCanal.soundTransform = miVolumen;
}
```

Esta función se encarga de aplicar, a la propiedad **volume** de la instancia **miVolumen**, el valor entre **0** y **1** que se haya generado en la subclase **Volume**. Puede que en principio resulte complejo comprender el modo en el que se comunican dos clases, pero su implementación, a medida que los proyectos las requieran, simplificará notoriamente el desempeño de nuestros desarrollos y delegaremos procesos a bloques de códigos. De este modo, en caso de querer aplicar cambios al control de volumen, simplemente trabajaremos sobre la clase **Volume** y no se modificará el resto del contenido.
A su vez, si bien el desarrollo de clases generalmente corre por cuenta de los desarrolladores, conocer su funcionamiento puede simplificarle muchísimo trabajo a quienes se estén iniciando en Flash ya que conociendo la sintaxis para aplicarlas pueden, con muy pocas líneas de código, reutilizar el trabajo realizado por otra persona sin necesidad de comprender el código que se encuentra dentro de la clase.

¿Qué son las etiquetas ID3?

Cuando se codifica un archivo MP3, los **encoders** pueden incluir **metadatos** dentro del archivo. Estos metadatos contienen información sobre el archivo de audio y se los conoce como **etiquetas ID3**. Flash nos permite conocer estos metadatos por medio de la clase **ID3Info**. A su vez, contamos con un evento que nos indica cuando se encuentran disponibles las etiquetas ID3:

```
miSonido.addEventListener(Event.ID3, onID3Info, false, 0, true);
```

Como vemos, este evento es aplicable a nuestro sonido, no a nuestro canal. Una vez que se propaga el evento, disponemos de toda la información ID3 que haya sido in-

dicada al codificarse el archivo de audio. Hay **siete metadatos** para los que Flash nos facilita la lectura por medio de propiedades. En nuestro ejemplo, sólo accederemos a la información referente al álbum y al artista del tema en reproducción:

```
private function onID3Info(event:Event):void{

     miSonido.removeEventListener(Event.ID3, onID3Info);
     var album:String = miSonido.id3.album;
     var artist:String = miSonido.id3.artist;

     infoTxt.appendText(" - album: " + album);
     infoTxt.appendText(" - artista: " + artist);
}
```

Las etiquetas ID3 están compuestas por cuatro caracteres y, como dijimos anteriormente, Flash nos proporciona propiedades para acceder a las siete más empleadas:

ETIQUETA ID3	PROPIEDAD
COMM	Sound.id3.comment
TALB	Sound.id3.album
TCON	Sound.id3.genre
TIT2	Sound.id3.songName
TPE1	Sound.id3.artist
TRCK	Sound.id3.track
TYER	Sound.id3.year

Tabla 2. *Propiedades de ActionScript 3.0 correspondientes a las etiquetas ID3 más utilizadas.*

La ventaja de estas propiedades es que, definitivamente, es más sencillo recordarlas. Por ejemplo, para acceder a la información de género al que pertenece una pista de audio, emplearemos la sintaxis **`Sound.id3.genre`** y no **`Sound.id3.TCON`**. Además de las mencionadas, en el sitio **http://livedocs.adobe.com/flash/9.0_es/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00002131.html** encontraremos un listado con todas las **etiquetas ID3** a las que podemos acceder desde Flash y una breve descripción de cada una de ellas.

¿Cómo leer todas las etiquetas ID3 disponibles para un MP3?

Creando una instancia de la clase **`ID3Info`**, se generará un objeto que almacenará toda la información disponible para un archivo de audio. Por medio de una sentencia **for..in**, podemos recorrer ese objeto y saber de qué etiquetas disponemos.

```
var id3Object:ID3Info = event.target.id3;
for (var propiedad:String in id3Object) {
     trace("etiqueta id3: ", propiedad, ":", id3Object[propiedad]);
}
```

Si bien las etiquetas ID3 pueden ser de gran utilidad, muchas veces no se hace uso de ellas. En los casos para los que no existe el metadato por el que estamos consultando, Flash nos devolverá **null**. Si necesitamos un metadato en particular, es recomendable usar una expresión regular para verificar que el metadato coincide con la información que estamos solicitando o bien que su valor no es nulo.

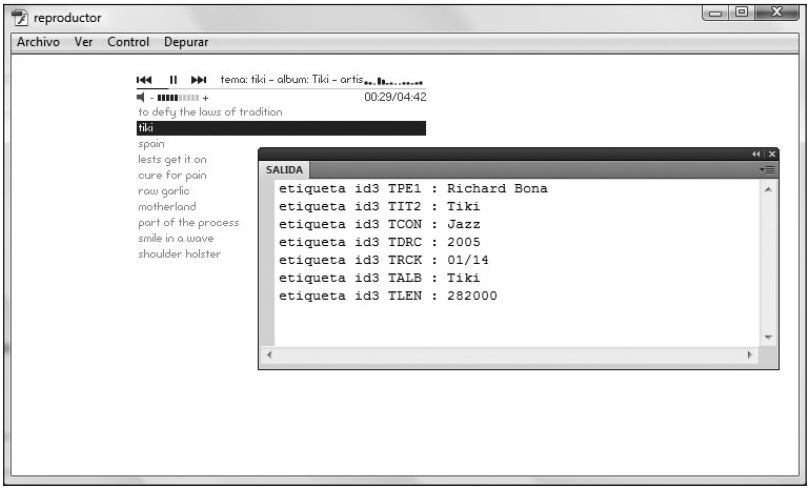


***Figura 6.** Etiquetas ID3 disponibles para un tema.*

¿Cómo modificar etiquetas ID3?

Las etiquetas ID3 son de **sólo lectura**, por lo que no podemos modificar su contenido en tiempo de ejecución desde Flash, pero sí es posible cambiar manualmente sus valores. Para eso, debemos ubicar en nuestro sistema operativo el MP3 cuyas etiquetas queremos modificar y, luego, hacer un clic con el botón derecho sobre él

WWW.ID3.ORG

En el sitio web **www.id3.org** encontraremos información respecto a las etiquetas ID3, sus características, su evolución y sus versiones. Allí también hallaremos las últimas novedades, y puede resultarnos de gran utilidad en caso de querer profundizar en el tema y conocer un mayor detalle respecto a estas etiquetas y a su implementación.

y elegir la opción **Propiedades...** del menú que aparece. En el cuadro que se abre, vamos a la pestaña **Detalles** y allí modificamos el valor que necesitamos cambiar.

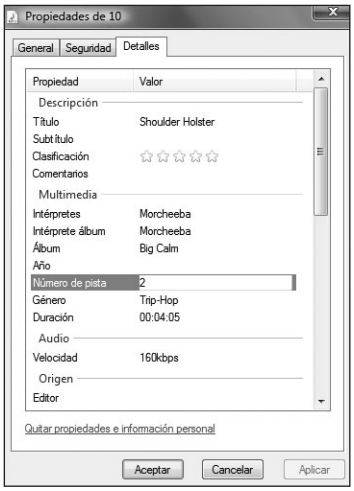


***Figura 7.** Dentro del panel de propiedades del archivo de audio, podemos modificar algunas etiquetas del MP3.*

Acceder al tiempo total y transcurrido de una pista de audio

Con las propiedades **length** de la clase **Sound** y **position** de la clase **SoundChannel**, que ya hemos visto anteriormente, podemos acceder al tiempo total y al transcurrido de un archivo de audio. Suponiendo que sólo necesitásemos el tiempo total, alcanzaría con leer la propiedad **length** de la clase **Sound**. Pero si lo que queremos hacer es indicar el tiempo transcurrido, debemos emplear un **timer** para que estos valores se actualicen continuamente.

No hemos utilizado timers hasta aquí, pero veremos que son muy fáciles de emplear y son de una gran utilidad. Al crear la instancia, podemos indicar dos parámetros: **new Timer(demora, repetir);**. En el parámetro **demora**, establecemos cada cuánto queremos que se ejecute el timer. Este valor se expresa en milisegundos y, como necesitamos que se actualice cada un segundo, definimos su valor en 1000 milisegundos. De este modo, esta función se ejecutará cada 1 segundo. En el parámetro **repetir**, podemos indicar cuántas veces queremos que se reitere esta operación. Al no definir ningún valor, el timer se ejecutará cada un segundo de forma indefinida. Por medio del método **start()** damos comienzo al conteo del timer:

```
myTimer = new Timer(1000);
myTimer.addEventListener(TimerEvent.TIMER, setTiempo, false, 0, true);
myTimer.start();
```

Con esta configuración, cada un segundo se llamará a la función **setTiempo()**, cuyo código podemos ver a continuación:

```
private function setTiempo(Event:TimerEvent):void{
     var totalTime:String = transformTime(miSonido.length);
     var parcialTime:String = transformTime(miCanal.position);
     timeTxt.text = parcialTime + "/" + totalTime;
}
```

Dentro de la función **setTiempo()**, debemos utilizar la propiedad **length** de la clase **Sound** y la propiedad **position** de la clase **SoundChannel** para conocer el tiempo total y el transcurrido de una pista de audio. Ambas propiedades están expresadas en milisegundos y esto no nos es de utilidad si lo que queremos es expresar el tiempo total y el transcurrido en el formato 00:00 (minutos:segundos).
Para traducir los milisegundos a una cadena de texto en el formato que necesitamos, empleamos la función **transformTime**. Esta función recibe un valor numérico (en milisegundos), por medio de operaciones matemáticas traducimos esa cantidad de milisegundos al formato minutos y segundos, y la función lo devuelve en modo de cadena de texto (**String**).

```
private function transformTime(time:Number):String{
     var minutos:uint = time / 60000;
     var minutosSobra:uint= time%60000;
     var segundos:uint = minutosSobra/1000;
     var segundosString:String = (segundos.toString().length==1) ? "0" +
segundos.toString() : segundos.toString();
     var minutosString:String = (minutos.toString().length==1) ? "0" + mi
nutos.toString() : minutos.toString();
     var setPosition:String = minutosString + ":" + segundosString;
     return setPosition;
}
```

Un minuto equivale a 60 segundos y un segundo se fragmenta en mil milisegundos. Por lo tanto, 60 multiplicado 1000 nos dará la cantidad de milisegundos que contiene un minuto: 60000. Conociendo esta información, dividimos la cantidad total de milisegundos en 60000 y obtenemos la cantidad de **minutos**:

```
var minutos:uint = time / 60000;
```

El módulo **%** es una operación que nos permite conocer el **resto** de una división. Accediendo a ese resto, podemos obtener los milisegundos que no han llegado a conformar un minuto, veamos cómo usarlo:

```
var minutosSobra:uint= time%60000;
```

Dividiendo ese valor en 1000 (equivalente de los milisegundos que contiene un segundo), obtenemos los **segundos**, como vemos a continuación:

```
var segundos:uint = minutosSobra/1000;
```

Por último, controlamos por medio de condicionales si los minutos o los segundos son menores a **9** para que se agregue un **0** delante de la cadena.

```
var segundosString:String = (segundos.toString().length==1) ? "0" + segun
dos.toString() : segundos.toString();
var minutosString:String = (minutos.toString().length==1) ? "0" +
minutos.toString() : minutos.toString();
```

Una vez concluidas las operaciones, devolvemos por medio de **return** la cadena de texto, como vemos en la línea siguiente:

```
var setPosition:String = minutosString + ":" + segundosString;
return setPosition;
```

Figura 8. *Traducción a minutos y segundos (00:00) de la duración (**Sound.length**) y la posición (**SoundChannel.position**) de un archivo de audio.*

Barras de tiempo transcurrido y streaming

Para finalizar con nuestro reproductor le agregaremos dos funcionalidades más: una barra para el **progreso** del tema y otra que indique la **carga** de los contenidos. Las acciones necesarias para ambas barras las haremos desde un evento **ENTER_FRAME**:

```
addEventListener(Event.ENTER_FRAME, setBars, false, 0, true);
```

Para generar una barra que nos indique el progreso de la reproducción, simplemente dividimos la posición actual por el largo total de la pista de audio, y el resultado de dicha división se lo aplicamos a la propiedad **scaleX** del sprite **timeLine**.

```
var playbackPercent:Number = miCanal.position / miSonido.length;
timeLine.scaleX = playbackPercent;
```

En cuanto a la barra que nos indica el porcentaje de tema cargado, la sintaxis es muy similar, sólo que debemos dividir los bytes cargados por los bytes totales del MP3. Para acceder a esta información, lo hacemos por medio de las propiedades **bytesLoaded** y **bytesTotal** de la clase **Sound**.

```
if(streamLine.scaleX <1){
     var soundLoaded:Number = miSonido.bytesLoaded / miSonido.bytesTotal;
     streamLine.scaleX = soundLoaded;
}
```

¿Cuándo emplear timers y cuándo el evento ENTER_FRAME?

Así como para generar el contador del tiempo transcurrido empleamos un **Timer**, para las barras de carga y progreso hemos utilizado un evento **ENTER_FRAME**. Si bien ambos tienen características y comportamientos diferentes, generalmente uno puede suplir las funciones del otro y viceversa. Esto no implica que sea correcto utilizar uno u otro sin haber hecho un análisis previo. Por el contrario, entender con claridad qué es lo que queremos lograr es lo que determina cuándo utilizar un **Timer** y cuándo el evento **ENTER_FRAME**.

En el ejemplo que estamos trabajando, para modificar el valor del tiempo transcurrido, utilizamos un **Timer**. La ventaja de este elemento es que definimos que se ejecute una vez por segundo. También podríamos haber utilizado un evento **ENTER_FRAME** y visualmente los resultados serían los mismos, pero mientras el evento **ENTER_FRAME** se ejecutaría 24 veces por segundo, el **Timer** lo hace una única vez por segundo. Como vemos, a nivel de optimización de recursos, el evento

ENTER_FRAME se estaría ejecutando 23 veces innecesariamente. Por este motivo, para ese caso en particular, un **Timer** es la mejor opción y cumple con nuestro cometido: actualizar nuestro tiempo transcurrido cada un segundo. Con las **barras de progreso** sucede todo lo contrario. Suponiendo que un tema dure tan sólo 10 segundos, con nuestro **Timer** veríamos 10 saltos (uno cada 10 segundos). En cambio, con un evento **ENTER_FRAME** vemos una barra continua.
En esta primera etapa, incorporamos los conceptos necesarios para cargar un sonido de manera externa, asignarle un canal, detener y reanudar su reproducción, conocer su duración total y su posición, modificar su volumen, acceder a sus etiquetas ID3 y conocer los bytes cargados y los bytes totales del tema. A partir de ahora, abarcaremos un nuevo concepto en el desarrollo en ActionScript: el acceso a los datos de un sonido a fin de crear representaciones visuales.

DATOS DE SONIDO Y REPRESENTACIONES VISUALES

Desde la salida de ActionScript 3.0, contamos con el método **computeSpectrum()** de la clase **SoundMixer**, por medio de la cual podemos leer los datos de las formas de onda del sonido que se está reproduciendo en el momento. Ésta es una novedad interesante y muy esperada porque, conociendo esta información, es posible crear representaciones visuales de los sonidos que se están reproduciendo. A la información que podemos acceder es:

- La amplitud de un sonido.
- La frecuencia de un sonido.

A su vez, por medio de las propiedades **rightPeak** y **leftPeak** de la clase **SoundChannel**, conoceremos la **amplitud** de los canales izquierdo y derecho de un audio.

Acceder a la amplitud de los canales

La amplitud de los canales es la información a la que podemos acceder y manipular de forma más sencilla. Mediante las propiedades **leftPeak** y **rightPeak** del objeto **SoundChannel**, es posible conocer la amplitud de cada canal estéreo que se reproduce. Los valores para estas propiedades van del **0**, que es el **silencio**, al **1**, que es la **amplitud total**. Aplicando esta información a las propiedades de un **Sprite**, podemos generar de una forma muy sencilla la representación visual de ambos canales, como veremos en el ejemplo que vamos a realizar.

Figura 9. *Información visual de los canales de la pista de audio; accedemos por medio de las propiedades rightPeak y leftPeak.*

En primer lugar, creamos dos **Sprites**; uno para el canal derecho y otro para el canal izquierdo. Para lograrlo, usamos el siguiente código:

```
leftChannel = new Sprite();
leftChannel.scaleY = leftChannel.alpha = 0;
leftChannel.y = 380;
leftChannel.graphics.beginFill(0xffffff, 1);
leftChannel.graphics.drawRect(0, 0, 130, 150);
leftChannel.graphics.endFill();
mainContainer.addChild(leftChannel);

rightChannel = new Sprite();
rightChannel.scaleY = rightChannel.alpha = 0;
rightChannel.x = 140;
rightChannel.y = 380;
rightChannel.graphics.beginFill(0xffffff, 1);
rightChannel.graphics.drawRect(0, 0,130, 150);
rightChannel.graphics.endFill();
mainContainer.addChild(rightChannel);
```

Como veremos a continuación, aprovechamos el evento **ENTER_FRAME** que estamos utilizando en nuestro código para asignar los valores de los canales a las propiedades **scaleY** y **alpha** de nuestros **Sprites**:

```
leftChannel.scaleY = -miCanal.leftPeak;
leftChannel.alpha = miCanal.leftPeak;
rightChannel.scaleY = -miCanal.rightPeak;
rightChannel.alpha = miCanal.rightPeak;
```

Lo interesante de este proceso es que su implementación es realmente muy fácil, y con unas pocas líneas de código es posible crear una representación visual de los canales del audio. A continuación veremos el método **computeSpectrum**, que requiere de una mayor complejidad, pero nos permite generar efectos más interesantes que los que obtuvimos hasta ahora.

Método computeSpectrum

Por medio del método **computeSpectrum**, podemos obtener los datos de un sonido. Esta información es devuelta como un objeto **ByteArray** que contiene 512 datos; cada uno de estos es un valor de **coma flotante** entre -1 y 1. Estos valores son los que representan la amplitud de los puntos de la onda de sonido de una reproducción. De los 512 datos que contiene el **ByteArray**, los primeros 256 pertenecen al canal izquierdo y los 256 restantes al canal derecho.

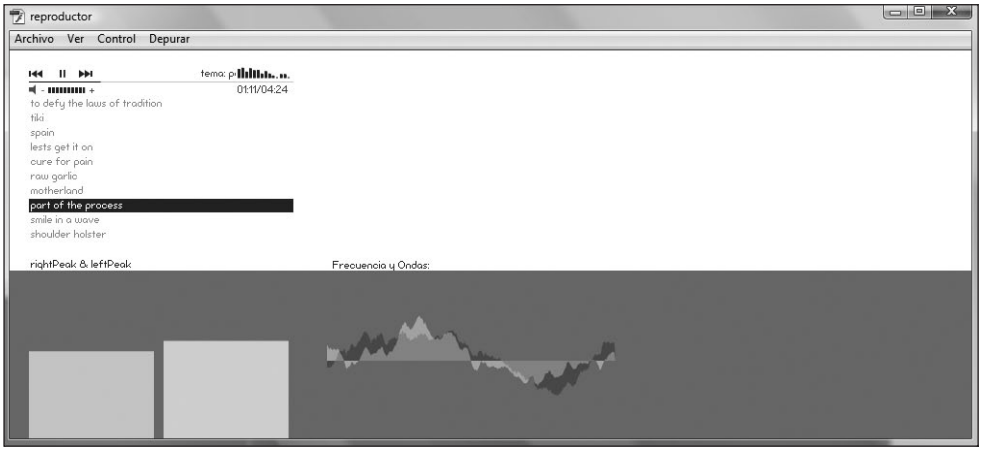


Figura 10. *No debemos confundir los gráficos: el primero responde a la amplitud de los canales y el segundo es una representación gráfica de los puntos de la onda de sonido (método* ***computeSpectrum****).*

FFT: Fast Fourier Transform

Cuando empleamos el método **computeSpectrum** podemos establecer el parámetro **FFTMode** (**Fast Fourier Transform**), por medio del cual se convierten los datos de forma de onda en datos de espectro de frecuencias.

Figura 11. *En el modo de visualización de frecuencia (FFT) podemos generar gráficos que representen la frecuencia del audio.*

Generar representaciones visuales

Insistiendo con la importancia de las clases, éste es un claro ejemplo de los beneficios de su implementación. Crearemos una clase a la que llamaremos **WaveVisualization** y, dependiendo de los parámetros que le indiquemos, nos permitirá obtener diversos resultados visuales de nuestros sonidos. La clase **WaveVisualization.as** contiene todo el código encargado de realizar la representación visual del espectro de sonido, pero la debemos llamar desde nuestro código, la clase **Mp3Player**:

```
var waveObject:Object = new Object();
waveObject.altoEspectro = 100;
waveObject.anchoEspectro = 300;
waveObject.setFFT = true;
waveObject.dibujarRelleno = false;
waveObject.canalIzquierdoColor = 0xcccccc;
waveObject.canalDerechoColor = 0x555555;

wave = new WaveVisualization(waveObject);
wave.x = 330;
wave.y = 170;
addChild(wave);
```

En primer lugar, creamos un objeto que nombraremos **waveObject** y contendrá todos los parámetros que le enviaremos a la clase (veremos la importancia de estos parámetros más adelante). Una vez que generamos el constructor de la clase **WaveVisualization**, le pasamos el objeto como parámetro, lo posicionamos sobre las coordenadas **x** e **y**,

y agregamos los contenidos a la lista de visualización. Luego, dentro de la clase **WaveVisualization**, una vez recibido el objeto, debemos asignar los valores de éste a las variables que tenemos dentro de la clase:

```
public function WaveVisualization(myObject:Object):void {

     miSprite = new Sprite();
     addChild(miSprite);

     altoEspectro = myObject.altoEspectro || 150;
     canalIzquierdoBase = myObject.canalIzquierdoBase || 150;
     canalDerechoBase = myObject.canalDerechoBase || 150;
     canalDerechoColor = myObject.canalDerechoColor || 0xff066;
     canalIzquierdoColor = myObject.canalIzquierdoColor || 0xff0066;
     setFFT = myObject.setFFT || false;
     canales = myObject.canales || "both";
     dibujarLinea = myObject.drawLine || false;
     dibujarRelleno = myObject.drawFill || true;
     anchoEspectro = (myObject.visualWidth / 256)|| 1;
     addEventListener(Event.ENTER_FRAME, definirCanales, false, 0, true);
}
```

Una de las ventajas de pasar como parámetro un objeto con los valores que hayamos definido es que no debemos preocuparnos por el orden de las variables. Simplemente, enviamos el objeto y desde la clase accedemos a sus valores. Podríamos haber pasado los valores de las variables como parámetros, pero tendríamos que recordar el orden en el que deben ir. Enviando un objeto, accedemos directamente a sus propiedades sin preocuparnos por el orden. Esto es especialmente útil en este tipo de clases en la que podemos agregar nuevos parámetros. A su vez, definimos una serie de valores por defecto en caso de que no sean asignados al enviar el objeto:

```
canalIzquierdoColor = myObject.canalIzquierdoColor || 0xff0066;
setFFT = myObject.setFFT || false;
```

En caso de que no se establezca el color para la visualización de las ondas del canal izquierdo, definimos **0xff0066** como valor por defecto. Del mismo modo, en caso de que no indiquemos que la visualización será aplicando el parámetro **FFT**, ésta se omitirá. Por último, una vez que tenemos asignados todos los pares de valores, iniciamos el evento **ENTER_FRAME** y llamamos a la función **definirCanales**.

```
private function definirCanales(event:Event):void {
SoundMixer.computeSpectrum(miByteArray, setFFT);
     miSprite.graphics.clear();
     if(canales == "both"){
            dibujarData(canalIzquierdoColor, canalIzquierdoBase);
            dibujarData(canalDerechoColor, canalDerechoBase);
     }else if(canales == "left"){
            dibujarData(canalIzquierdoColor, canalIzquierdoBase);
     }else{
            dibujarData(canalDerechoColor, canalDerechoBase);
     }
}
```

Ésta es una parte fundamental de nuestro código. En primer lugar, definimos el método **computeSpectrum**, como vemos a continuación:

```
SoundMixer.computeSpectrum(miByteArray, setFFT);
```

Observamos que en el primer parámetro incluimos la instancia de nuestro **ByteArray** y en el segundo, el valor booleano **setFFT**. Luego, hacemos uso del método **clear()** de la API de dibujo de Flash (en caso de no recordar la sintaxis, podemos recurrir al **capítulo 3**, totalmente dedicado al dibujo en Flash):

```
miSprite.graphics.clear();
```

Necesitamos borrar el dibujo dado que la función **definirCanales** se va a estar ejecutando constantemente por medio del evento **ENTER_FRAME**; de no hacerlo, cada vez que ingresemos a esta función se superpondrán los dibujos de las ondas constantemente, y no es esto lo que queremos. Por último, en función del valor

III CAPABILITIES PARA EL MANEJO DE AUDIO

Independientemente de la propiedad **hasMP3** que tiene la clase **Capabilities**, ésta cuenta con una serie de propiedades que pueden ser de utilidad a la hora de desarrollar aplicaciones de audio. En este sentido, aconsejamos averiguar sobre **hasAudio**, **hasAudioEncoder** y **hasStreamingAudio**.

de la variable **canales**, sabemos si dibujar ambos canales, sólo el izquierdo o sólo el derecho. Los valores que aceptamos para esta variable son las cadenas de texto **"ambos"**, **"izquierdo"** y **"derecho"**. El valor por defecto que definimos en nuestro caso es **"ambos"**, por lo tanto se dibujarán los dos canales:

```
if(canales == "ambos"){
     dibujarData(canalIzquierdoColor, canalIzquierdoBase);
     dibujarData(canalDerechoColor, canalDerechoBase);
}
```

Vemos que estamos llamando a la función **dibujarData()**, que es la encargada de generar el espectro de sonido. A esta función le pasamos dos parámetros, el color del canal y la posición de base que tendrá:

```
private function dibujarData(color:uint, base:Number):void {
     if(dibujarLinea) miSprite.graphics.lineStyle(0, color, 1, true);
     if(dibujarRelleno) miSprite.graphics.beginFill(color, 0.3);
     miSprite.graphics.moveTo(0, base);
     for (var i:uint = 0; i<256; i++) {
            miSprite.graphics.lineTo(i* anchoEspectro, base - (miByteA
rray.readFloat() * altoEspectro));
     }
     miSprite.graphics.lineTo((i* anchoEspectro) , base);
     miSprite.graphics.endFill();
}
```

Primero averiguamos, condicional mediante, si definimos que se vean las líneas de las formas (**dibujarLinea**), sus rellenos (**dibujarForma**) o ambas cosas. Recordemos que estos son dos valores booleanos y de ser verdaderos, definimos el estilo de línea (método **lineStyle** de la API de dibujo) y después, el relleno para la forma (método **beginFill**).

III PODCASTING

Éste es un concepto relativamente nuevo en la Web 2.0. Un **podcast** es un archivo **RSS** mediante el que se distribuyen contenidos de audio, generalmente en formato **MP3** o **ACC**. Al tratarse de archivos **RSS** (**estructuras XML**), Flash nos permite su lectura. Con algunas pequeñas modificaciones al código de nuestro reproductor de MP3, podemos generar un reproductor de **podcasts**.

```
if(dibujarLinea) miSprite.graphics.lineStyle(0, color, 1, true);

if(dibujarRelleno) miSprite.graphics.beginFill(color, 0.3);
```

Luego, empleamos el método **moveTo()** que, como hemos visto en el **capítulo 3**, equivale a posicionarnos en una coordenada (x e y) desde la que comenzaremos a dibujar. Anteriormente, hemos hecho la analogía de este método con lo que sería bajar el lápiz para comenzar a escribir:

```
miSprite.graphics.moveTo(0, base);
```

Para movernos a un punto, debemos hacerlo por medio del método **lineTo()**, indicando las coordenadas **x** e **y**. A la posición **x**, nos desplazamos multiplicando el valor de **i** por el ancho que hayamos definido para el espectro, y para la posición **y** accedemos al método **readFloat()** de nuestro **ByteArray** y lo multiplicamos por el alto:

```
for (var i:uint = 0; i<256; i++) {
     miSprite.graphics.lineTo(i* anchoEspectro, base - (miByteArray.read
Float() * altoEspectro));
}
```

Lo que hace el método **readFloat()** es leer el siguiente valor dentro del **ByteArray**. Anteriormente, dijimos que el **ByteArray** contiene 512 valores, por lo que al hacer un bucle **for** del 0 al 256 recorreremos la mitad de estos valores dibujando el espectro para el canal izquierdo y volvemos a dibujar 256 valores para el canal derecho.
Un detalle que debemos tener en cuenta y que tiende a confundir es que el bucle **for** recorre de 0 a 256, pero no vemos un bucle **for** que lo haga del 256 al 512. Recordemos que este bucle se ejecuta dos veces; una para dibujar el espectro del canal izquierdo y otra para el derecho. Lo que hace que se dibujen ambos canales es, justamente, el método **readFloat()**. Independientemente de que el bucle **for** recorra valores del 0 al 256, el valor que en verdad nos proporciona la onda del audio es la posición a la que accedemos por medio del método **readFloat()**, por lo que al ingresar por primera vez al bucle **for** el valor de **readFloat()** será de 0 a 255 para el canal izquierdo y, al volver a ingresar al bucle **for**, independientemente de que el **for** vuelva a recorrer del 0 al 255, estamos accediendo a los valores que van del 256 al 512 dentro de nuestro **ByteArray**: de este modo, el primer **for** recorre los primeros 256 valores y el segundo, los 256 restantes: del 256 al 512. Una vez recorridos todos los valores y creadas las líneas para nuestro espectro, creamos la última línea y cerramos nuestro dibujo.

```
miSprite.graphics.lineTo((i* anchoEspectro) , base);
miSprite.graphics.endFill();
```

Ventajas de nuestro visualizador de ondas y frecuencias

Haber trabajado nuestro código del modo en que lo hicimos tiene tres ventajas principales. Por un lado, la **escalabilidad** del código, ya que al tenerlo dentro de una **clase**, cualquier modificación que queramos hacer la efectuamos sobre esa clase sin afectar el rendimiento de nuestro reproductor de audio. Por otro lado, mencionamos la **sencillez de su implementación**, ya que si bien podríamos hacer que nuestra clase alcance una gran complejidad, su implementación es verdaderamente sencilla, como podemos ver en el siguiente código:

```
var waveObject:Object = new Object();
waveObject.altoEspectro = 100;
waveObject.anchoEspectro = 300;
waveObject.canalIzquierdoColor = 0xffffff;
waveObject.canalDerechoColor = 0x555555;

wave = new WaveVisualization(waveObject);
wave.x = 330;
wave.y = 170;
addChild(wave);
```

Esto es beneficioso tanto para el desarrollador como para el diseñador que, conociendo lo básico e indispensable del manejo de clases, puede aplicar un desarrollo que genera espectros de sonido. Con menos de diez líneas de código tenemos una representación visual de nuestros sonidos.

Por último, es importante mencionar que la modalidad en la que desarrollamos nuestra clase nos permite una **flexibilidad** enorme. Con sólo modificar algunos parámetros, podemos alterar el resultado final y hacerlo funcional casi para cual-

III MÚSICA EN LA WEB

Las ventajas que nos brinda la carga externa de sonidos hacen de Flash el entorno ideal para desarrollar aplicaciones que reproducen audio. Un claro ejemplo de esto es el reproductor de MP3 que poseen las cuentas de usuario de **myspace**. Flash se consolidó como una plataforma útil y flexible a la hora de generar reproductores de audio, desplazando a otras tecnologías.

quier interfaz y no únicamente en lo que respecta al uso del parámetro **FFT** (**Fast Fourier Transform**). Veamos algunos ejemplos de lo que podemos lograr.

Con los siguientes parámetros, dibujamos líneas para nuestro espectro:

```
var waveObject:Object = new Object();
waveObject.altoEspectro = 100;
waveObject.anchoEspectro = 300;
waveObject.dibujarLinea = true;
waveObject.canalIzquierdoColor = 0xffffff;
waveObject.canalDerechoColor = 0x555555;
```

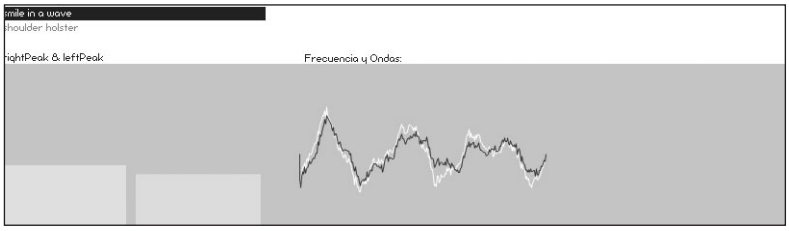


Figura 12. *Sólo dibujamos las líneas del espectro y la representación visual cambia completamente.*

Agregando un parámetro, los resultados son completamente distintos:

```
var waveObject:Object = new Object();
waveObject.altoEspectro = 100;
waveObject.anchoEspectro = 300;
waveObject.setFFT = true;
waveObject.dibujarLinea = true;
waveObject.canalIzquierdoColor = 0xffffff;
waveObject.canalDerechoColor = 0x555555;
```

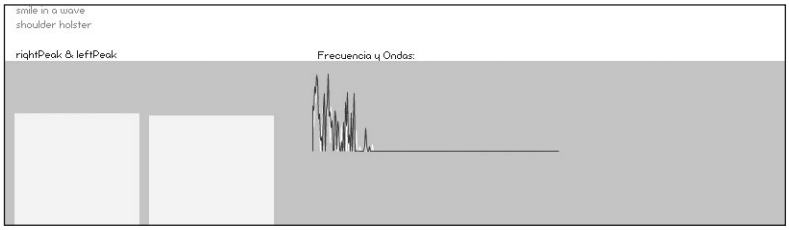


Figura 13. *Aplicando el parámetro FFT los resultados de nuestro espectro son diferentes.*

O bien rellenando las formas y sin dibujar las líneas obtenemos otros resultados. También podemos modificar el color de los canales, el alto, el ancho, y generar las combinaciones necesarias para adaptar el espectro de sonido a nuestras necesidades; muchas veces debemos adecuar el desarrollo del espectro a lo que el reproductor de MP3 nos pida. Otra alternativa podría ser que al cliquear el espectro, éste cambie su forma.
Dentro del paquete **visualizations** se encuentra la clase **FrequencyVisualization**. No tiene sentido que la analicemos en profundidad en vistas a que su lógica es exactamente la misma que la que utilizamos en la clase **WaveVisualization**. La única diferencia es que, en lugar de generar las formas del espectro de manera continua, empleamos una serie de barras para la representación visual de los sonidos.

```
var dataObject:Object = new Object();
dataObject.anchoBordes = 1;
dataObject.colorFormas = 0x333333;
dataObject.numeroBarras = 14;
dataObject.anchoBarra = 2;
dataObject.separacionBarras = 2;
vis = new FrequencyVisualization(dataObject);
vis.x = 237;
vis.y = 18;
addChild(vis);
```

Al igual que la clase que analizamos anteriormente, le pasamos como parámetro un objeto que contiene toda la información. Podemos definir el ancho de los bordes, el color de las barras, la cantidad de barras que tendrá, su ancho y la separación en píxeles entre ellas. Conociendo y manipulando estos valores, es posible generar una cantidad interesante de alternativas y adaptarlas al diseño que sea necesario.

```
private function dibujarData(event:Event):void {
SoundMixer.computeSpectrum(byteArray, true);
     container.graphics.clear();
     container.graphics.lineStyle(1, colorFormas, 1);
     container.graphics.beginFill(colorFormas,1);
     for (var i:int = 0; i<numeroBarras; i++) {
            offset = (intervaloBarras * byteArray.readFloat());
            container.graphics.drawRect( (i*anchoBarra)*separacionBarras,
(intervaloBarras - offset), anchoBarra, offset);
     }
}
```

Por último, la función **dibujarData()** se encarga de crear los contenidos constantemente. Al igual que en nuestra clase anterior, definimos el método **computeSpectrum**, le asignamos el **ByteArray** y establecemos en **true** el parámetro **FFT**. Luego, borramos el último dibujo por medio del método **clear()** de la API de dibujo, definimos el estilo de línea por medio del método **lineStyle()** y el relleno de las formas con el método **beginFill()**. La única diferencia considerable con la clase que explicamos anteriormente es que, al generar los dibujos, no recorremos los 256 valores de los canales sino la cantidad de barras que hayamos definido:

```
for (var i:int = 0; i<numeroBarras; i++) {
```

Dentro del bucle que ingresamos, leemos la información del **ByteArray** por medio del método **readFloat()** y, a través del método **drawRect()**, generamos los rectángulos que darán forma a nuestra representación visual:

```
offset = (intervaloBarras * byteArray.readFloat());
container.graphics.drawRect( (i*anchoBarra)*separacionBarras, (intervaloBa
rras - offset), anchoBarra, offset);
```

Con un poco de ingenio y conocimientos matemáticos, es posible lograr maravillosos resultados en las representaciones visuales que llevemos a cabo. El método **computeSpectrum** nos permite dotar de un mayor realismo a nuestras aplicaciones. Años atrás, para simular representaciones, generalmente se desarrollaban gráficos con movimientos aleatorios. Hoy en día, gracias al potencial de este método, podemos tener acceso, en tiempo real, a lo que está sucediendo con la amplitud, la frecuencia y las ondas del sonido. Las posibilidades que esto nos brinda son muchísimas.

... RESUMEN

Conocer las clases involucradas en el manejo de sonidos nos brinda los recursos necesarios para poder desarrollar aplicaciones robustas. Como vimos a lo largo del capítulo, saber implementar el método computeSpectrum() de la clase SoundMixer pone a nuestra disposición una herramienta muy potente si buscamos generar representaciones visuales de los sonidos. Conocer e implementar estos recursos nos permite crear aplicaciones de mayor impacto. En el próximo capítulo nos adentraremos en el uso de video en Flash.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1) ¿Cuáles son las ventajas de cargar sonidos de manera externa?

2) ¿Qué clases están involucradas en el manejo de sonido?

3) ¿Qué método de la clase **Sound** utilizamos para comenzar a reproducir un sonido?

4) ¿Cómo se asigna un canal a un sonido?

5) ¿Cómo podemos definir el tiempo de almacenamiento en buffer de un tema?

6) ¿Qué son las etiquetas ID3? ¿Qué clase utilizamos para leerlas?

7) ¿Qué propiedad de la clase **SoundChannel** indica la posición en la que se encuentra una pista de sonido?

8) ¿Qué clase necesitamos para aplicar transformaciones al volumen o al paneo de una pista de audio?

9) ¿Qué es la clase **SoundMixer**?, ¿qué método de ésta utilizamos para detener todos los sonidos?

10) ¿Qué propiedades de la clase **Sound** nos brindan información respecto a los bytes cargados y a los bytes totales de una pista?

EJERCICIOS PRÁCTICOS

1) Cree su propio reproductor de audio.

2) Agregue un botón que le permita al usuario hacer que la reproducción de los temas sea de manera aleatoria.

3) Agregue un botón que, al haber sido presionado, repita constantemente un tema una vez que finaliza su reproducción.

4) Acceda a todas las etiquetas ID3 de las que dispongan las pistas de audio.

5) Cree su propio visualizador de frecuencias de sonido.

Video en Flash

Desde que Adobe creó el formato FLV de video para Flash, éste se convirtió en uno de los recursos más utilizados a la hora de implementar videos en la Web. Su versatilidad, la facilidad de manejo y la relación entre calidad y peso lo posicionaron como el formato de preferencia. Basta con ver las páginas web más relevantes de videos en la red y notaremos que todas trabajan con formato FLV o MP4.

SERVICIO DE ATENCIÓN AL LECTOR: usershop@redusers.com

INTRODUCCIÓN A FLASH VIDEO: FLV

El **FLV** (**Flash Video**) es un formato que ha sido diseñado para ver videos en Internet, empleando **Flash Player**. Si bien es cada vez más común que los usuarios tengan videos en formato FLV en sus equipos, si queremos hacer uso de ellos en la Web, debemos incrustarlos dentro de un archivo SWF, ya sea como archivos internos o realizando una carga externa. El principal beneficio de este formato es que para que los usuarios puedan reproducir un video solamente necesitan tener instalado Flash Player en su computadora. A su vez, nos brinda una relación interesante entre la **calidad de video** y su **peso**. Estos son los dos motivos principales por los cuales el formato FLV se ha consolidado como la principal alternativa para implementar videos en la Web. A continuación, veremos de qué manera crear videos en este formato, cómo importarlos a Flash y cómo cargarlos de manera externa.

Generar videos FLV

En caso de querer crear un video en formato FLV, podemos hacerlo mediante el software **Adobe Media Encoder CS4**, programa que nos permite **codificar** videos de cualquier formato en FLV. A esta aplicación podemos acceder desde la IDE de Flash, o bien desde nuestro sistema, y está disponible para su instalación en cualquier paquete de la suite de **Adobe CS4**. En este capítulo, simplemente veremos de qué modo codificar cualquier video al formato FLV.

Figura 1. *En el sitio* ***http://help.adobe.com/es_ES/AdobeMediaEncoder/4.0/WS053EA898-B158-4c20-8147-FE0881119BE2.html*** *encontraremos una guía completa para el uso de este programa.*

■ Generar un video FLV

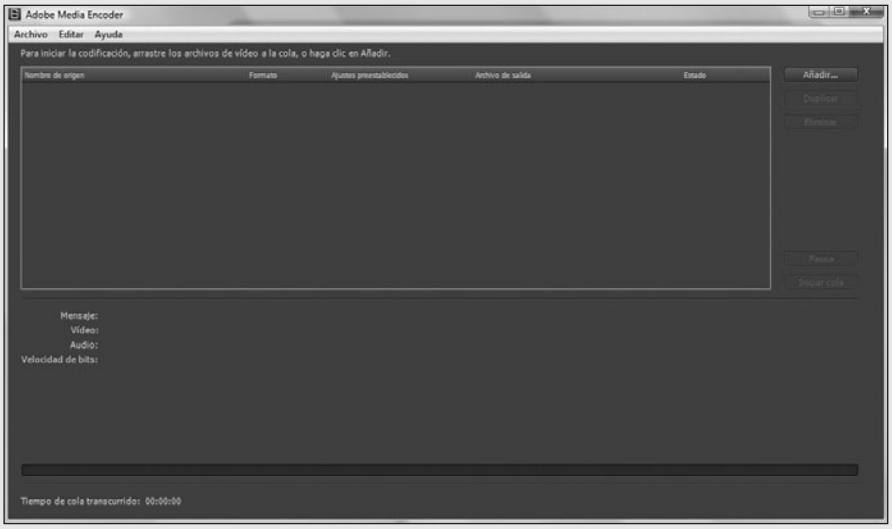
PASO A PASO

1 Una vez dentro de Flash, diríjase a **Archivo/Importar** y seleccione **Importar vídeo...**.

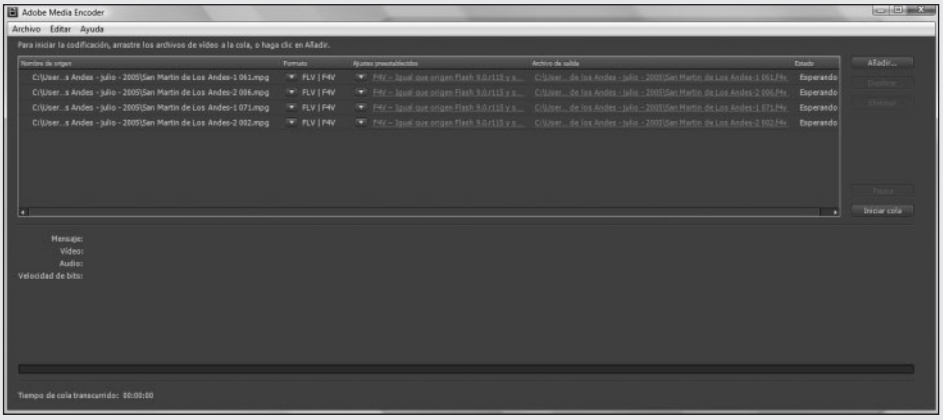
2 Dentro de la ventana **Importar vídeo**, presione **Iniciar Adobe Media Encoder**.

3 En caso de no haber guardado el archivo antes de comenzar, Flash abrirá un cuadro de diálogo para que lo haga. Guarde el archivo en su sistema operativo.

4 Una vez almacenado el archivo, accederá a **Adobe Media Encoder**. Su interfaz es realmente fácil de usar. Puede arrastrar los videos y soltarlos sobre la cola o bien presionar el botón **Añadir...** y buscar los videos que quiere **codificar**.

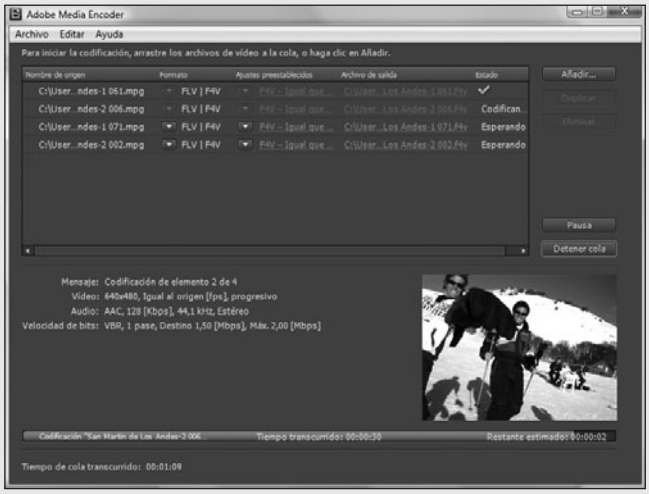


5 Cuando añada los videos a la cola, podrá ver una serie de parámetros como **Nombre de origen**, **Formato**, **Ajustes preestablecidos**, **Archivo de salida** y **Estado**. Por defecto, los videos FLV se guardarán en el mismo lugar en el que se encuentran los archivos de origen. Si quiere aplicar algún cambio (en el ajuste, en la ruta de salida o en el formato), debe hacerlo desde aquí, antes de comenzar con el proceso de codificación.



6 Luego de agregar todos los archivos necesarios y de realizar las modificaciones que desee, presione el botón **Iniciar cola**.

7 Comenzado el proceso, verá una barra de progreso y la información relativa al video, al audio y a la velocidad de bits. Cuando finaliza, contará con los archivos FLV listos para utilizar.



Importar videos en formato FLV

Desde la **IDE** de Flash podemos importar videos **FLV**. Si bien Flash acepta otros formatos (**.AVI**, **.MPG**, **.MPEG**, **.WMV**, **.ASF**), el que nos interesa y que ha dotado de un enorme potencial a la plataforma es el formato FLV, al cual dedicaremos este capítulo. A continuación veremos cómo importar videos dentro de una película SWF y, luego, cómo cargarlos de manera externa.

■ Importar video FLV a Flash — PASO A PASO

1 Una vez dentro de Flash, diríjase a **Archivo/Importar** y seleccione **Importar vídeo...**.

2 Dentro de la nueva ventana **Importar vídeo**, hay dos posibilidades: puede seleccionar un video que se encuentre en su equipo, que es la opción por defecto, o utilizar uno que haya sido subido a la Web. En caso de que el video se encuentre en su equipo, presione el botón **Examinar...** para ubicarlo e importarlo. En caso de que el video se encuentre en la Web, debe tildar la opción **Ya se ha implementado en un servidor Web, Flash Video Streaming Service o Flash**. Esto habilitará el campo de introducción de texto que se encuentra debajo. Ingrese la **URL** del video en ese campo.

III INTEGRACIÓN DE SOFTWARE DE ADOBE PARA VIDEO

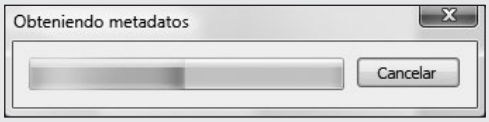
En caso de realizar **postproducción**, tanto en **Adobe Premiere Pro CS4** como en **Adobe After Effects CS4** es posible exportar videos en formato **FLV**. Ésta es una de las principales ventajas de contar con una **suite de Adobe** que nos permita hacer uso de estos programas en conjunto.

3 Una vez seleccionado el archivo de video, elija entre las opciones **Cargar vídeo externo con componente de reproducción** o **Incorporar FLV en SWF y reproducir en línea de tiempo**. Flash recomienda emplear la primera opción y utilizar la segunda sólo para videos cortos, debido a problemas con la sincronización del sonido.

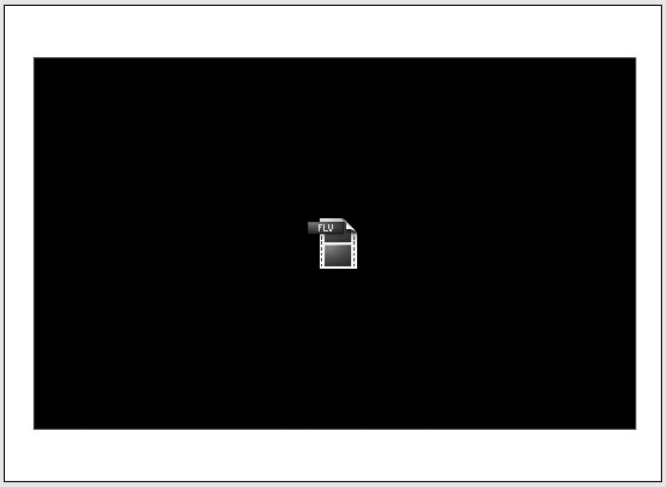
4 Al presionar **Siguiente**, aparece la pantalla **Aplicación de aspectos**. Desde allí puede seleccionar qué tipo de configuración quiere darle al video. Básicamente, los aspectos son una serie de alternativas predeterminadas que Flash pone a su disposición para definir la **apariencia** del reproductor de video. Más adelante dedicaremos parte de este capítulo a los aspectos mediante el componente **FLVPlayback**. Por el momento, no defina ningún aspecto. Así importará únicamente el video en formato FLV sin su reproductor.

5 Al volver a presionar **Siguiente**, aparece **Finalizar importación de vídeo**. En este paso verá toda la información referente al video que se va a importar.

6 Haga clic nuevamente en **Siguiente** y observe el progreso de importación.



7 Finalizada la carga, el video aparecerá en la interfaz, ubicado en el escenario.



8 Al exportar la película, verá la reproducción del video.

Siguiendo los pasos anteriores, hemos podido incluir un video en formato FLV dentro de nuestro archivo FLA. Al exportar el FLA y generar una película SWF, ésta contendrá un video FLV. Como estos términos tienden a confundirse, el mejor modo de considerar este proceso es pensando que si bien FLV es un formato de video, para que funcione en la Web debe encontrarse dentro de un archivo SWF, ya que, como mencionamos antes, su ventaja principal es que el usuario sólo necesita tener Flash Player instalado para visualizarlo. En definitiva, cuando accedemos a un video en Internet, no estamos viendo un video en formato FLV sino una película SWF, que contiene dentro un video en formato FLV que ha sido importado de la manera en la que acabamos de hacerlo, o bien que se ha cargado de forma externa como veremos en las próximas páginas.
Por otro lado, cabe mencionar que Adobe incluye un nuevo software en su suite: **Adobe Media Player**. Si bien su instalación es opcional, es un buen modo de administrar nuestros videos **FLV** para poder visualizarlos, eliminarlos, generar listas de favoritos, etcétera. Su uso es bastante sencillo y puede ser de gran utilidad.

{ } CRECIMIENTO Y ACEPTACIÓN DEL FORMATO FLV

Una muestra del constante crecimiento y la aceptación que tuvo el formato **FLV** en el mercado es el uso que hacen de él los principales sitios de la Web 2.0. **YouTube**, **facebook**, **myspace**, **Google Video**, **Yahoo! Video**, entre otros, utilizan el formato **FLV** o **MP4** para los videos de sus sitios.

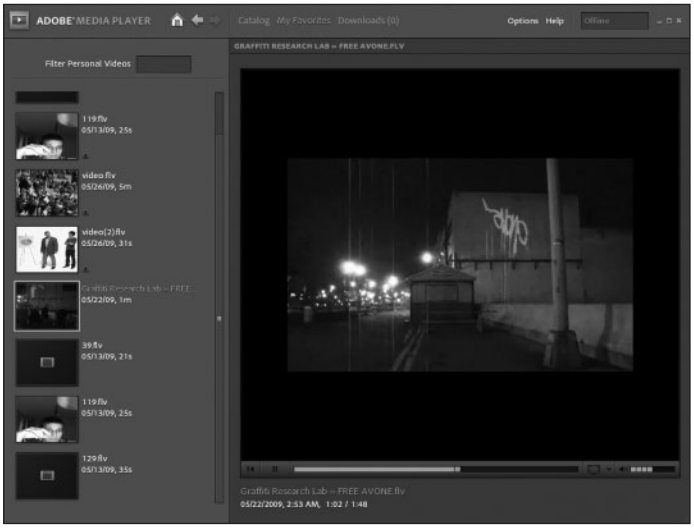


Figura 2. *Adobe Media Player nos permite organizar de manera eficiente los videos que se encuentran dentro de nuestro equipo.*

Emplear componentes: FLVPlayback

Los **componentes** son **colecciones** que han sido generadas previamente y que podemos implementar sin necesidad de llevar a cabo ningún desarrollo. Si bien están para que hagamos uso de ellos, utilizarlos tiene sus puntos a favor y en contra. Es un tema que ha generado opiniones divergentes y tendrá siempre sus defensores y detractores. Debido a esto, si no nos resulta de interés, podemos evitar esta sección ya que en las próximas páginas llevaremos a cabo el desarrollo de nuestro propio reproductor de video, creando sus controles y comportamientos.

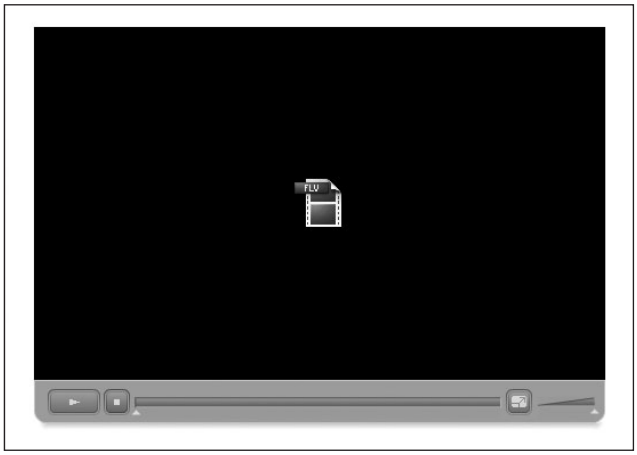


Figura 3. *Una de las características del componente* ***FLVPlayback*** *es que podemos definir, previamente, los controles de video con los que contará.*

La principal **ventaja** de utilizar componentes es que podemos contar con una aplicación funcional que no requiere de la escritura de código para su desarrollo; simplemente empleamos unas pocas líneas para su implementación, como podremos ver a continuación. En este caso, con pasos sencillos, podemos obtener un reproductor de video funcional con sus respectivos controles y no habremos invertido tiempo en su desarrollo. Su uso es sencillo e intuitivo para aquellos que recién comienzan con Flash, y no se requieren de conocimientos avanzados de ActionScript para realizar su implementación.
La gran **desventaja** de los componentes es que no son lo suficientemente personalizables como para obtener resultados interesantes, por lo que su apariencia es poco atractiva. Probablemente algunos desarrolladores consideren esto un tema menor, pero aquellos que conocen en profundidad la industria multimedial sabrán que el diseño es una parte fundamental en cualquier desarrollo, y un componente no parece ser lo más viable si lo que se busca es generar nuevas experiencias para el usuario. Otra desventaja importante es que incrementan considerablemente el peso final del archivo.
A modo de conclusión, podemos establecer que los componentes son completamente útiles si nuestras exigencias son medias o bajas y si la optimización de recursos no es un tema a tener en cuenta durante el desarrollo. Por otro lado, podemos afirmar que los componentes no son la opción indicada, si, como desarrolladores, creemos que el diseño es un valor agregado y consideramos que la optimización de recursos es una parte fundamental de cualquier proyecto.

Componente FLVPlayback

El componente **FLVPlayback** lo podemos aplicar sobre videos importados dentro de nuestra película o sobre videos externos. Para videos importados, una vez que definimos el archivo que vamos a cargar y llegamos a la opción **Aplicación de aspectos**, podemos elegir del menú desplegable el componente que mejor se adapte a nuestras necesidades. Cuando finalizamos el proceso, contaremos con el video y sus respectivos controles sobre el escenario.
De todos modos, no siempre tendremos que importar los videos, podemos cargarlos de forma externa, al igual que sucede con las imágenes y los sonidos. También es po-

III MÁS USUARIOS CON EL PLUGIN DE ADOBE FLASH PLAYER

Considerando que para poder visualizar videos en formato **FLV** es necesario contar con el plugin de **Adobe Flash Player**, los principales sitios de videos, con **YouTube** a la cabeza, son los mayores responsables del incremento en los índices de penetración de **Adobe Flash Player** en los últimos años.

sible utilizar el componente **FLVPlayback** en este caso. En primer lugar, debemos incluirlo dentro de nuestra **BIBLIOTECA**. Para hacerlo, abrimos el panel de **COMPONENTES** desde el menú **Ventana/Componentes** o presionando **CTRL+F7**, la ventana mostrará los elementos que posee Flash por defecto. Para incorporar el componente dentro de nuestro archivo, debemos arrastrarlo desde su panel hasta la biblioteca. De este modo, quedará dentro de nuestra película y podremos hacer uso de él.

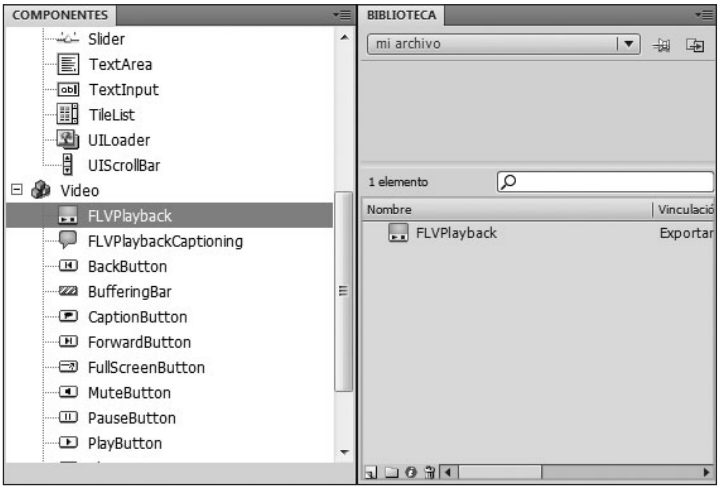


***Figura 4.** Para incluir el componente **FLVPlayback** en nuestro archivo, debemos arrastrarlo a la **BIBLIOTECA**.*

Para hacer la carga externa de un video utilizando este componente, debemos emplear unas pocas líneas de código, como vemos a continuación:

```
import fl.video.*;
var miVideo:FLVPlayback = new FLVPlayback();
miVideo.source = “ejemplo.flv”;
addChild(miVideo);
```

Para manipular un componente mediante código, tenemos que **importarlo**. Esto lo hacemos en la primera línea de nuestro código:

```
import fl.video.*;
```

Luego, creamos una instancia para nuestro componente, a la que llamamos **miVideo**, y por medio de la propiedad **source** indicamos la URL del video que vamos a cargar. Finalmente, añadimos la instancia al escenario por medio del método **addChild()**.

Figura 5. *Al exportar la película, el video de nuestro ejemplo se carga de manera externa.*

Por ahora, nuestro reproductor no cuenta con ningún control. Lo que hicimos fue aplicar las líneas básicas para poder importar el video. Si queremos incluir algún sistema de controles en el reproductor, debemos agregarlo. En primer lugar, necesitamos acceder al **INSPECTOR DE COMPONENTES**. En caso de no contar con él sobre la interfaz, tenemos que abrirlo presionando **MAYÚS + F7**, o bien desde **Ventana/Inspector de componentes**. Una vez que lo abrimos, arrastramos nuestro componente **FLVPlayback** desde la **BIBLIOTECA** al escenario y, al hacer un clic sobre él, veremos en el **INSPECTOR DE COMPONENTES** toda la información referente a éste.

Figura 6. *El* ***INSPECTOR DE COMPONENTES*** *nos permite modificar aspectos del componente* ***FLVPlayback*** *como su* ***skin*** *(aspecto), su* ***color****, su* ***transparencia*** *o* ***volumen****, entre otros.*

Para modificar el skin (aspecto), presionamos sobre su valor y aparecerá una lupa. Al hacer clic sobre ésta, se abrirá una nueva ventana que contiene todos los skins que podemos aplicarle a nuestro componente.

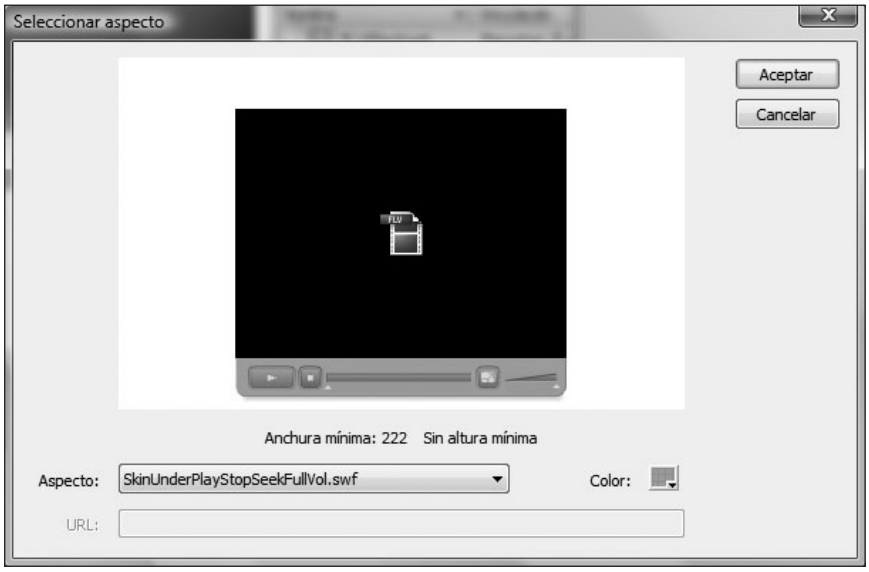


Figura 7. *Desde la pantalla* ***Seleccionar aspecto*** *podemos definir el* ***skin*** *y su* ***color****.*

Seleccionar un aspecto generará un archivo .SWF, que se guardará en la misma carpeta que contiene nuestro desarrollo. Ese archivo .SWF es, justamente, el skin para nuestro reproductor. Para aplicarlo, debemos hacerlo por medio de la propiedad **skin** del componente **FLVPlayback**. Con estos sencillos pasos contamos con un reproductor de videos FLV con sus respectivos controles.

```
import fl.video.*;
var miVideo:FLVPlayback = new FLVPlayback();
miVideo.source = “ejemplo.flv”;
miVideo.skin = “SkinUnderPlayStopSeekFullVol.swf”
addChild(miVideo);
```

DESARROLLAR UN REPRODUCTOR DE VIDEO

Desde la llegada de ActionScript 3.0, se han incluido una serie de novedades y mejoras que vale la pena que analicemos. El mejor modo de conocer estas novedades y los comportamientos de un reproductor de video es llevando a cabo su

desarrollo. A continuación crearemos nuestro propio reproductor con sus respectivos controles y exploraremos todo lo referente al uso de video en Flash. Una vez concluida esta parte del capítulo, contaremos con un reproductor de video con una importante cantidad de funcionalidades.
Ya hemos visto en capítulos anteriores las dificultades que acarrea contar con los archivos dentro de nuestra película, pero al tratarse del video, el problema es aún mayor. Si bien el peso de un video puede variar de acuerdo a la duración, a las dimensiones y a la calidad, generalmente estos archivos ocuparán un espacio considerablemente mayor que el de las imágenes o los archivos de audio. Es por este motivo que al trabajar con videos, es altamente recomendable hacerlo con archivos externos a nuestra película y hacer una carga progresiva de sus contenidos.

Figura 8. *El resultado final de nuestro reproductor de video.*

A diferencia del capítulo anterior en el cual creamos una clase principal que contiene dentro un reproductor, en este caso nuestra clase principal no contendrá el código del reproductor sino una llamada a la clase **VideoPlayer**, que es la que se encarga de generar nuestro reproductor de video.

III CRECIMIENTO DE YOUTUBE

YouTube vio la luz en febrero de 2005 y fue desarrollado por tres ex empleados de la compañía **PayPal**. En octubre de 2008, **Google** adquirió el sitio de videos **YouTube** por 1.650 millones de dólares, convirtiéndose en una de las compras más grandes de un sitio hasta el momento.

***Figura 9. Flujo de datos**: desde la clase principal (**MainClass**) le enviamos a nuestro reproductor (**VideoPlayer**) la ruta del video a reproducir.*

La principal ventaja de desarrollar nuestro reproductor de este modo es que al implementarlo, simplemente, tenemos que crear una instancia de éste y pasarle los parámetros que sean necesarios. Generaremos una clase a la que llamaremos **MainClass.as** e incluiremos dentro de ella el código que se encuentra a continuación. Debemos vincular esta clase al archivo FLA, ya que será nuestra clase principal:

```
package com{

     import flash.display.Sprite;
     import com.VideoPlayer;

     public class MainClass extends Sprite {

            private var miVideoPlayer:VideoPlayer;

            public function MainClass():void {
                   init();
            }

            private function init():void {
                   miVideoPlayer = new VideoPlayer();
                   miVideoPlayer.x = Math.round(stage.stageWidth/2 - miVi
deoPlayer.width/2);
                   miVideoPlayer.y = Math.round(stage.stageHeight/2 - miVi
deoPlayer.height/2);
                   miVideoPlayer.loadVideo("videos/video.f4v");
                   addChild(miVideoPlayer);
            }
     }
}
```

En primer lugar, debemos importar nuestra clase que genera el reproductor:

```
import com.VideoPlayer;
```

Luego, debemos crear una instancia de la clase que acabamos de importar y posicionarla sobre las coordenadas **x** e **y**:

```
miVideoPlayer = new VideoPlayer();
miVideoPlayer.x = Math.round(stage.stageWidth/2 - miVideoPlayer.width/2);
miVideoPlayer.y = Math.round(stage.stageHeight/2 - miVideoPlayer.height/2);
```

Por último, para cargar un video, todo lo que tenemos que hacer es emplear el método **loadVideo** que definimos dentro de la clase **VideoPlayer**:

```
miVideoPlayer.loadVideo("videos/video.flv");
addChild(miVideoPlayer);
```

Por medio del método **loadVideo()**, le pasamos como parámetro la ruta del video a cargar a la clase **VideoPlayer**. Lo que vimos hasta aquí es necesario, pero el código interesante que contiene el desarrollo de nuestro reproductor de video es la clase **VideoPlayer**, cuyos aspectos fundamentales veremos a continuación.

Conexión entre el cliente y el servidor: clase NetConnection

Antes de comenzar a reproducir un video, debemos establecer una conexión entre Flash y el servidor. Para hacerlo, tenemos que emplear el objeto **NetConnection**, que crea un **canal** (**cliente-servidor**).

```
private function establecerConexion():void{
     nc = new NetConnection();
     nc.connect(null);
}
```

Es importante saber que por medio de **NetConnection** solamente estamos generando un canal y estableciendo una conexión, pero no se transfieren datos. Para transferir datos a través del canal debemos utilizar el objeto **NetStream** que veremos un poco más adelante. Mientras, recordemos que por medio del método **loadVideo** indicábamos, desde la clase **MainClass**, la ruta del video a cargar:

```
miVideoPlayer.loadVideo("videos/video.flv");
```

Desde la clase **VideoPlayer** recibimos la cadena con la ruta, la igualamos a una variable que emplearemos dentro de la clase y, luego, llamamos a la función **conectarStream()**.

```
public function loadVideo(rutaVideo:String):void{
     this.rutaVideo = rutaVideo;
     conectarStream();
}
```

La función **conectarStream()** es la que contiene el código más importante de nuestro desarrollo, y lo analizaremos en detalle a continuación:

```
private function conectarStream():void {

     stream = new NetStream(nc);
     stream.bufferTime = 5;
     stream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler,
false, 0, true);

     var client:Object = new Object();
     client.onMetaData = onMetaData;

     stream.client = client;
     video.attachNetStream(stream);
     stream.play(rutaVideo);

     addEventListener(Event.ENTER_FRAME, loadBar, false, 0, true);
     addEventListener(Event.ENTER_FRAME, progressBar, false, 0, true);
}
```

Como hemos dicho anteriormente, por medio de **NetConnection** sólo creamos un canal; para transferir contenidos, debemos utilizar la clase **NetStream**. La clase **NetStream** es un canal dentro del objeto **NetConnection** que generamos. Esta clase cuenta con una cantidad considerable de métodos y propiedades mediante las cuales podemos controlar tanto el progreso de carga de los contenidos como la reproducción. A lo largo de este capítulo, los iremos conociendo y trabajaremos con ellos para familiarizarnos con su forma de utilización.

En principio, para crear una instancia de la clase **NetStream**, debemos asignarle al constructor el objeto **NectConnection** sobre el que queremos crear el canal:

```
stream = new NetStream(nc);
```

Recordemos que llamamos **nc** a la instancia del objeto **NetConnection**. Una vez creada la instancia de la clase **NetStream**, podemos acceder y manipular prácticamente todos los datos necesarios para contar con un reproductor de video.

Almacenar el video en el buffer

En el capítulo anterior vimos cómo utilizar el **buffer** para realizar la carga de sonidos externos. Mediante este almacenamiento nos aseguramos de que la cantidad de tiempo que indiquemos se reproduzca sin interrupción. Así como para definir el buffer en el sonido lo hacíamos mediante la clase **SoundLoaderContext**, al trabajar con video nos resultará más sencillo; simplemente le asignamos la cantidad de segundos para almacenar mediante la propiedad **bufferTime** de la clase **NetStream**.
Es importante recordar que en el caso de los sonidos, a la clase **SoundLoaderContext** debemos indicarle el tiempo de almacenamiento en buffer en **milisegundos** y a la propiedad **bufferTime**, en **segundos**. Tenemos que ser cuidadosos con esto ya que podríamos cometer errores importantes; no es lo mismo asignar 5 segundos al buffer de video que confundirnos y asignarle 5000.

```
stream.bufferTime = 5;
```

De este modo, nos aseguramos de que los primeros 5 segundos del video se cargarán antes de comenzar la reproducción y que se visualizarán sin interrupciones.

Mostrar el video cargado

Por sí sola, la clase **NetStream** no puede mostrar los datos que está cargando. Para reproducir un video, debemos hacer un **attach** de los datos a una instancia de éste.

III FORMATOS DE VIDEOS ACEPTADOS POR LA CLASE NETSTREAM

Desde la versión 9.0.115.0 de **Flash Player**, podemos reproducir archivos derivados del formato **MPEG-4**. Esto implica que no sólo es posible cargar archivos **FLV** sino también aquellos con extensiones **F4V**, **M4A**, **MP4**, **M0V**, **MP4V**, **3GP** y **3GP2**. Este formato nos ofrece una excelente calidad de video. Los videos en alta definición de YouTube no se almacenan en formato FLV, sino en MP4.

Al comenzar nuestro código, dentro de la función **crearGUI()**, generamos una instancia del objeto **video** como vemos a continuación:

```
video = new Video();
container.addChild(video);
```

La clase **Video** es la que se ocupa de mostrarnos el video que se está cargando de forma progresiva. Para poder reproducirlo, en primer lugar debemos asignarle la instancia de la clase **NetStream** que estamos utilizando por medio del método **attachNetStream()**. Esto lo hacemos de la siguiente manera:

```
video.attachNetStream(stream);
```

El método **attachNetStream()** indica que el video que se está cargando por medio de **NetStream** se muestre dentro de los límites de la instancia de video que creamos. Podríamos considerar a la instancia del video como un **contenedor** dentro del cual reproducimos el video y que podemos manipular como cualquier objeto de la lista de visualización. Veremos que es posible modificar su ancho, su alto y su transparencia, entre otras cosas. Para comenzar la reproducción, lo hacemos por medio del método **play()**, al que debemos indicarle la ruta del video como vemos en la siguiente línea. Recordemos que esa información la almacenamos en la variable **rutaVideo**.

```
stream.play(rutaVideo);
```

Es importante notar que la instancia del objeto **video** tiene una cantidad considerable de propiedades que nos permiten manipularlo, modificando su escala, su transparencia, la rotación, el ancho, el alto, etcétera.

Tamaño de videos

Si bien por medio del objeto **video** podemos modificar las dimensiones del video que vamos a reproducir, tenemos que ser cuidadosos con los valores que asignamos. Los videos generalmente deben respetar determinados formatos y si no aplicamos las transformaciones a conciencia, conservando las escalas sobre el alto y el largo, éste tiende a romperse. No está mal asignar nuevas medidas a un video, pero hay que hacerlo conociendo sus dimensiones nativas, a fin de aplicar **transformaciones proporcionales**. En nuestro ejemplo, no modificaremos las propiedades **width** y **height**; solamente reproduciremos videos de 320 px x 240 px.

Información sobre estados: evento NetStatusEvent

Los objetos **NetConnection** y **NetStream** distribuyen objetos **NetStatusEvent**. Este evento nos informa respecto al estado de un video. Para acceder a esta información debemos emplear la constante **NET_STATUS**.

```
stream.addEventListener(NetStatusEvent.NET_STATUS, getEstados, false, 0,
true);
```

El evento nos devuelve un objeto denominado **info**, que describe el estado o un error. La siguiente tabla presenta las propiedades más importantes y una breve descripción de cada una de ellas. Además, en la dirección **http://help.adobe.com/es_ES/AS3LCR/Flash_10.0/flash/events/NetStatusEvent.html** encontraremos un listado con todas las propiedades.

PROPIEDAD	SIGNIFICADO
"NetConnection.Connect.Closed"	Recibimos este mensaje cuando una conexión se ha cerrado de forma correcta.
"NetConnection.Connect.Success"	Se ha establecido una conexión de forma correcta.
"NetStream.Buffer.Empty"	El buffer se encuentra vacío. Debemos aguardar a que se llene nuevamente para que se reanude la reproducción (ver siguiente propiedad).
"NetStream.Buffer.Full"	El buffer está nuevamente lleno y va a reanudar la reproducción.
"NetStream.Play.Start"	Se inició la reproducción.
"NetStream.Play.Stop"	Se detuvo la reproducción.
"NetStream.Play.StreamNotFound"	No se encuentra el archivo que se le ha pasado como parámetro al método **play()**.
"NetStream.Connect.Closed"	Se ha cerrado de forma correcta la conexión.
"NetStream.Connect.Failed"	Ha fallado el intento de conexión.
"NetStream.Connect.Success"	La conexión se ha establecido correctamente.

***Tabla 1.** Información disponible a través del evento NetStatusEvent. Próximamente veremos cómo utilizarla.*

Seguramente, ningún proyecto que implique el manejo de video requiera usar toda esta información disponible, pero es bueno saber que contamos con ella y

III PROPIEDADES WIDTH Y HEIGHT, VIDEOWIDTH Y VIDEOHEIGHT

Es importante distinguir estos cuatro atributos del objeto **video**, ya que no representan lo mismo: **width** y **height** son propiedades de lectura y escritura, lo que implica que asignándoles valores podemos definir las dimensiones del reproductor. Las propiedades **videoWidth** y **videoHeight** son de sólo lectura, e indican las dimensiones reales del video que se está reproduciendo.

conocer sus características para cuando la necesitemos. Para acceder a los estados que nos brinda la clase **NetStatusEvent**, debemos aplicarle su correspondiente Event Listener a **netStream** o a **netConnection**:

```
stream.addEventListener(NetStatusEvent.NET_STATUS, getEstados, false, 0,
true);
```

Y para saber el código que nos devuelve el evento, podemos averiguarlo por medio de la sentencia **switch**, como vemos a continuación:

```
private function getEstados(event:NetStatusEvent):void {
     switch (event.info.code) {
```

En nuestro ejemplo solamente utilizaremos la información necesaria para tener un control sobre las actividades principales de nuestro reproductor. Por medio del código **"NetStream.Play.Start"** sabemos que se inició la reproducción. Esta información nos sirve para enviar el cabezal del clip llamado **toggle** (para el stop y el play) al frame **3** y mostrar el signo de detener la reproducción; a su vez, definimos la variable **isPlaying** en **true** para saber si el video se está reproduciendo o no.

```
case "NetStream.Play.Start" :
     trace("playing");
     isPlaying = true;
     toggle.gotoAndStop(3);
break;
```

Con el código **"NetStream.Play.Stop"** sabemos que se ha detenido la reproducción.

```
case "NetStream.Play.Stop" :
     trace("stop...");
     isPlaying = false;
break;
```

Detener y reanudar la reproducción

Por medio del método **togglePause()** de la clase **NetStream**, podemos detener o reanudar la reproducción. La primera vez que lo llamamos se realiza una pausa y al volver a llamarlo se reanuda la reproducción:

```
toggle.addEventListener(MouseEvent.MOUSE_DOWN, setTogglePause, false, 0,
true);

private function setTogglePause(event:MouseEvent):void{
     stream.togglePause();
     if(isPlaying){
            isPlaying = false;
            toggle.gotoAndStop(2);
     }else{
            isPlaying = true;
            toggle.gotoAndStop(3);
     }
}
```

Figura 10. *Alternamos el clip* ***toggle*** *entre el icono de* ***pausa*** *y el de* ***play*** *para que el usuario pueda detener o reanudar el video. Realizamos esta acción por medio del método* ***togglePause()****.*

Controlar el volumen de los videos

El funcionamiento del sonido en los videos es igual que para el audio. La clase **NetStream** cuenta con la propiedad **soundTransform**, a la que le debemos indicar el sonido del video en un valor entre **0** (silencio) y **1** (volumen máximo). Recordemos que para definir el volumen, tenemos que utilizar la clase **SoundTransform**. Del siguiente modo, estaríamos asignando el volumen máximo para nuestro video:

```
var st:SoundTransform = new SoundTransform();
st.volume = 1;
stream.soundTransform = st;
```

Y de esta manera estaría en **mute** (silencio):

```
var st:SoundTransform = new SoundTransform();
st.volume = 0;
stream.soundTransform = st;
```

Como veremos, para nuestro reproductor de videos utilizaremos exactamente el mismo control de volumen que implementamos en el capítulo anterior cuando trabajamos con nuestro reproductor de MP3. La ventaja de emplear una clase para ese caso la vimos en el capítulo anterior, pero de algún modo este ejemplo lo ratifica. El haber empleado ese elemento hace que no tengamos que pensar en el desarrollo del slider para el volumen; independientemente de que sea un reproductor de MP3 o un reproductor de video.
Entonces, simplemente creamos la instancia de nuestra clase y sabemos que ella nos devolverá un valor entre **0** y **1**. Lo único que resta por hacer es aplicárselo al sonido, ya sea de un audio o de un video. Veamos, en el código que aparece a continuación, como debemos realizarlo:

```
miVolume = new Volume(0x555555, 0xdddddd);
miVolume.setParams(1);
miVolume.addEventListener(Volume.VOLUME_SETTER, volumeSetter, false, 0,
true);
miVolume.y = contenedorBarras.y + 8;
container.addChild(miVolume);

private function volumeSetter(event:Event):void{
     var st:SoundTransform = new SoundTransform();
     st.volume = miVolume.returnVol;
     stream.soundTransform = st;
}
```

DETECTAR EL FINAL DE UN VIDEO

Así como podemos saber la posición en segundos de un video, también es posible conocer su duración total. No se usan condicionales para detectar el final de un video; el valor de la propiedad **time**, una vez terminada la reproducción, no siempre es el mismo que el metadato que contiene el tiempo total (**duration**). Utilizamos los eventos **NetStatusEvent** para averiguar la finalización de un video.

Figura 11. *El control de volumen, situado en el borde inferior izquierdo del reproductor de video.*

Conocer el tiempo transcurrido de un video

Por medio de la propiedad **time** de la clase **NetStream**, accedemos al tiempo transcurrido en segundos de un video. Podemos utilizar esta información para crear un **contador** que nos indique el lapso de reproducción de un video:

```
private function progressBar(event:Event):void{
     var temp:Number = stream.time;
     var minutes:int = temp / 60;
     var seconds:int = temp % 60;
     tiempoTxt.text = ((minutes<=9) ? "0"+ minutes.toString() :
minutes.toString()) + ":" + ((seconds<=9) ? "0" + seconds.toString() : se
conds.toString());
     barraTiempo.width = stream.time / videoTiempo * barraFondo.width;
}
```

{ } RELACIÓN DE ASPECTO (ASPECT RATIO) DE LOS VIDEOS

El aspect ratio **4:3** ha sido el estándar más utilizado en la Web, pero, desde noviembre de 2008, **YouTube** cambió el aspect ratio del reproductor de su sitio al formato **16:9**. Éste se utiliza principalmente para el formato de televisores plasma, y es lo que se conoce como **widescreen**.

Ya hemos visto una sintaxis muy similar en el capítulo anterior a fin de determinar el tiempo transcurrido en nuestro reproductor de MP3. Para el contador de tiempo de este capítulo, la gran diferencia es que la información que nos devuelve la propiedad **time** de la clase **NetStream** está expresada en segundos y no en milésimas como lo hace la propiedad **time** de la clase **Sound**. Por este motivo, para obtener los minutos no dividimos el tiempo total en 60000 (milisegundos que hay dentro de un minuto) sino en 60:

```
var temp:Number = stream.time;
var minutes:int = temp / 60;
```

De este modo, traducimos a minutos los segundos transcurridos de reproducción. En caso de que quede un resto de segundos que no conformen un minuto, los obtenemos conociendo el resto por medio del módulo **%**:

```
var seconds:int = temp % 60;
```

Por último, en caso de que los minutos o segundos sean menores o iguales a 9, agregamos un 0 delante, como vemos en el siguiente código:

```
tiempoTxt.text = ((minutes<=9) ? "0"+ minutes.toString() : minutes.toS
tring()) + ":" + ((seconds<=9) ? "0" + seconds.toString() : seconds.toS
tring());
```

Puede resultar confuso a primera vista, pero no son, ni más ni menos, que dos condicionales **if**. Para los minutos, utilizamos lo siguiente:

```
(minutes<=9) ? "0"+ minutes.toString() : minutes.toString()
```

VARIACIONES EN LA DURACIÓN TOTAL DE UN VIDEO

En muchas oportunidades, el valor de la duración total de un video incluido en sus metadatos no es real sino una aproximación a ella. Por este motivo, cuando finalice la reproducción de un video, no siempre el valor de la propiedad **time** de la clase **NetStream** será igual al valor devuelto por la propiedad **duration** que utilizamos para acceder a los metadatos.

Para los segundos, en cambio, debemos ingresar lo que aparece a continuación:

```
(seconds<=9) ? "0" + seconds.toString() : seconds.toString());
```

A su vez, aprovechamos el evento **ENTER_FRAME** para escalar la barra **barraTiempo** en función del tiempo transcurrido. Para eso, usamos la siguiente línea:

```
barraTiempo.width = stream.time / videoTiempo * barraFondo.width;
```

Conocer la duración total de un video

Para acceder a la duración total de un video, debemos asignarle un objeto a la propiedad **client** de la clase **NetStream** en el que podamos controlar los datos de **flujo** de un archivo de video:

```
var client:Object = new Object();
client.onMetaData = onMetaData;
stream.client = client;
```

Como podemos imaginar, el objeto **metaData** detecta cuándo se reciben los metadatos del video, y lo utilizamos de la siguiente manera:

```
private function onMetaData(data:Object):void {
     videoTiempo = data.duration;
     var min:int = videoTiempo / 60;
     var seg:int = videoTiempo % 60;
     videoSegundos = min.toString() + ":" + seg.toString();
}
```

III OTROS METADATOS DISPONIBLES

La duración total de un video no es el único metadato al que podemos tener acceso. Una vez que lo recibimos, es posible obtener otro tipo de información, por ejemplo, los fotogramas por segundo (**data.framerate**) y las medidas reales del video (**data.width** y **data.height**).

Por medio de la propiedad **duration**, conocemos la extensión total del video. Aplicando los mismos procedimientos que utilizamos en el paso anterior para calcular el tiempo transcurrido, logramos traducir a minutos y segundos la duración total del video. En nuestro ejemplo no haremos uso de esta información, pero es interesante saber que contamos con ella ya que nos puede resultar de utilidad en algún proyecto. En caso de querer incluirla, simplemente creamos un campo de texto y aplicamos el valor de la variable **videoSegundos**.

Modificar la posición del video: método seek()

Por medio del método **seek()** modificamos la posición en la que se encuentra el cabezal de reproducción del video. Con este método podemos crear un slider que nos permita realizar la variación. En nuestro reproductor, modificaremos la posición del video al hacer clic sobre el **Sprite** que contiene las distintas barras:

```
contenedorBarras.addEventListener(MouseEvent.MOUSE_UP, seekTime, false, 0,
true);
function seekTime(event:MouseEvent):void{
     barraTiempo.width = mouseX;
     var pos:Number = (videoTiempo * barraTiempo.width) / barraFondo.width;
     stream.seek(pos);
}
```

Cambiar la posición es realmente sencillo: debemos recordar que la barra que indica el progreso temporal del video (**barraTiempo**) incrementa su largo en función del tiempo transcurrido. Entonces, si vamos a modificar la posición del video, lo primero que tenemos que hacer es variar el largo de la barra en cuestión:

```
barraTiempo.width = mouseX;
```

De esta manera, le asignamos como nueva medida la posición del mouse en X (**mouseX**), que es la ubicación en la cual se hace clic. Una vez que la barra tomó su nuevo largo, ya contamos con los recursos para modificar la posición del video. Por medio de una regla de tres simple calculamos la nueva ubicación: multiplicando el tiempo total del video (**videoTiempo**) por el nuevo largo de la barra (**barratiempo.width**) y luego dividiendo este valor por la representación del tiempo total (**barraFondo.width**), obtenemos la nueva posición:

```
var pos:Number = (videoTiempo * barraTiempo.width) / barraFondo.width;
```

Por último, le asignamos esta nueva posición al método **seek**:

```
stream.seek(pos);
```

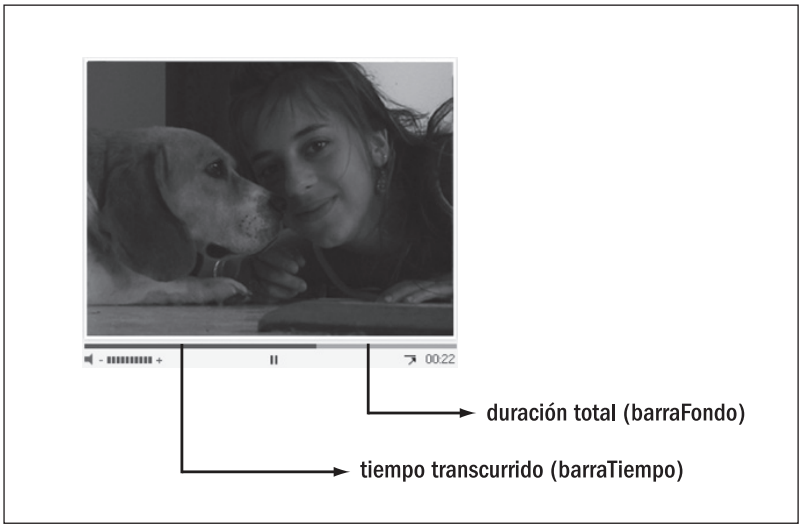


***Figura 12.** El sprite barraTiempo se escala en función del tiempo transcurrido. Haciendo clic sobre esta barra, podemos modificar su largo y, por lo tanto, el tiempo transcurrido del video.*

Averiguar bytes totales y bytes cargados de un video

A través de las propiedades **BytesLoaded** y **bytesTotal** de la clase **NetStream**, podemos acceder a los bytes que han sido cargados hasta el momento y a los bytes totales del video. Haciendo uso de estas dos propiedades, le informamos al usuario respecto del proceso de carga del video.

```
var videoLoaded:Number = stream.bytesLoaded/stream.bytesTotal;
barraStreaming.scaleX = videoLoaded;
```

Simplemente, dividiendo estos valores y aplicándoselos a la escala (**scaleX**) de un **Sprite**, podemos generar una barra que indique el progreso de la carga.

Hacer videos con fullscreen

Esta fue una de las grandes novedades de ActionScript 3.0: contar con la posibilidad de generar pantallas **fullscreen** (pantalla completa). Si miramos nuestra interfaz, en el borde inferior derecho, junto al tiempo, se encuentra un clip de película que, al presionarlo, generará una pantalla **fullscreen**. Una vez que este-

mos en modo pantalla completa, el icono cambiará de apariencia y se encargará de volver al modo normal cuando se haga clic sobre él nuevamente.

Figura 13. *El modo* ***fullscreen*** *ha sido una de las grandes novedades de ActionScript 3.0. A continuación veremos de qué manera implementarlo.*

En primer lugar, antes de comenzar con el código en Flash, debemos hacer una modificación sobre el archivo **HTML** que embebe nuestra película. Dentro de este tenemos que cambiar el valor del parámetro **allowFullScreen** y establecerlo en **true**. De este modo, el explorador permitirá el uso de la ventana completa.

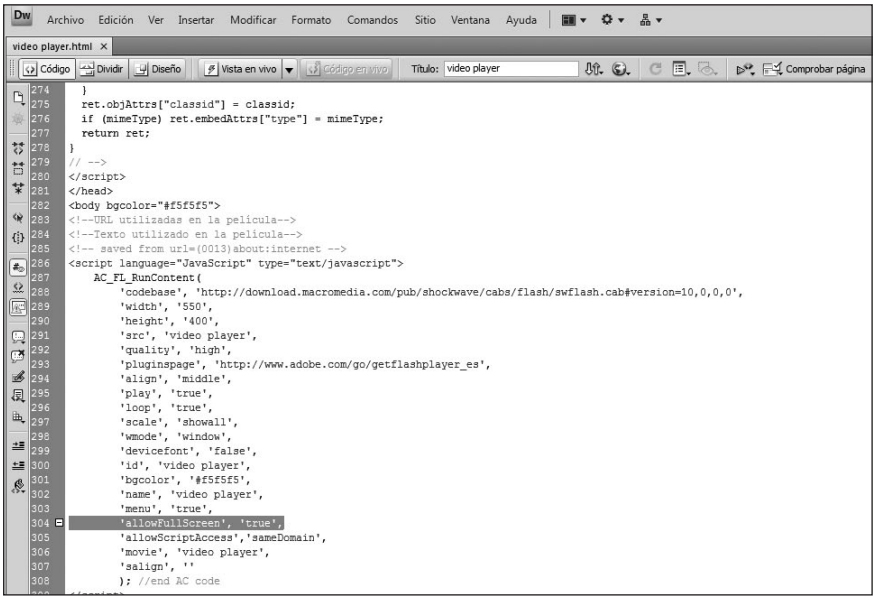


Figura 14. *Debemos modificar el valor de la propiedad* ***allowFullScreen*** *a* ***true*** *dentro del archivo* ***video player.html*** *para poder utilizar el modo pantalla completa en nuestro reproductor.*

Nuevamente, dentro de la clase de nuestro reproductor, en primer lugar, debemos asignar el Event Listener al botón que se encargará de generar el modo fullscreen, como podemos ver en el siguiente código:

```
fullScreen = new FullScreen();
fullScreen.addEventListener(MouseEvent.MOUSE_DOWN, setFullScreen, false, 0,
true);
fullScreen.x = 280;
fullScreen.y = 260;
container.addChild(fullScreen);
```

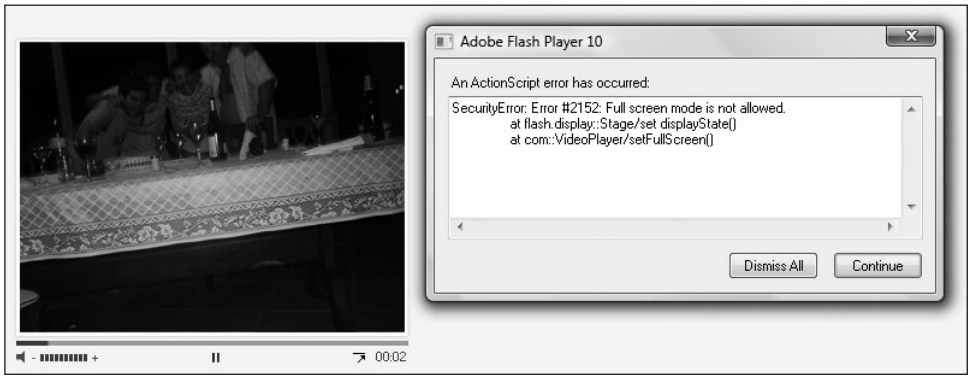


Figura 15. *En caso de no modificar el parámetro* ***allowFullScreen*** *dentro del archivo* ***HTML****, Flash generará un mensaje de error.*

Dentro de la función **setFullScreen()**, simplemente tenemos el siguiente código, que se encarga de cambiar el tamaño de la pantalla:

```
private function setFullScreen(event:MouseEvent):void{
     stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

SÓLO EL USUARIO PUEDE INICIAR EL MODO FULLSCREEN

El modo pantalla completa se ejecutará únicamente en respuesta a un clic de ratón o a la pulsación de una tecla por parte del usuario. En caso de querer generar una ventana **fullscreen** desde alguna parte del código y sin el consentimiento del usuario, Flash mostrará un error.

Por medio de la propiedad **displayState** del **Stage**, establecemos el modo pantalla completa. Para hacerlo, utilizamos la clase **StageDisplayState** y definimos la opción **FULL_SCREEN**. Eso es todo lo que necesitamos para generar el modo fullscreen.

Figura 16. *Nuestro reproductor en modo **fullscreen**. Para volver al modo normal, debemos presionar la tecla **Escape**.*

A su vez, Flash nos proporciona un evento por medio del cual podemos saber cuándo ingresamos en modo pantalla completa y cuándo salimos de él. Esto lo hacemos por medio del evento **FullScreenEvent**, que debemos asignarle al escenario (**stage**). El escenario no estará inmediatamente disponible al tratarse de una subclase (no olvidemos que la clase principal es **MainClass.as**) y, por este motivo, debemos detectar cuándo un elemento ha sido agregado al escenario. Esto lo podemos saber por medio de la constante **ADDED_TO_STAGE**:

```
container.addEventListener(Event.ADDED_TO_STAGE, onAdded, false, 0, true);

private function onAdded(event:Event):void{
     stage.addEventListener(FullScreenEvent.FULL_SCREEN, modoFullScreen,
```

DATOS IMPORTANTES AL INGRESAR AL MODO FULLSCREEN

Mientras nuestra película se encuentre en modo **pantalla completa**, se desactivarán todas las teclas, exceptuando aquellos métodos abreviados por medio de los cuales podemos salir de este modo. Generalmente, para salir, debemos presionar la tecla **ESCAPE**. Flash nos informará de esto por medio de un mensaje ubicado en el centro de la pantalla.

```
false, 0, true);
}
```

Escuchando al evento **FullScreenEvent**, sabemos cuándo estamos en modo fullscreen y cuándo no. En base a esto, asignamos y eliminamos listeners, tanto para entrar como para salir de este modo. Al comenzar la aplicación, siempre estará disponible el clip por medio del que se ejecuta el modo pantalla completa, pero una vez que se ejecutó, debemos eliminarle ese evento y asignarle otro por medio del que se pueda salir de este modo, tal como vemos en el siguiente código:

```
private function modoFullScreen(event:FullScreenEvent):void{
trace("se detecto el evento");
     if (event.fullScreen){
            trace("esta en fullscreen");
            fullScreen.removeEventListener(MouseEvent.MOUSE_DOWN, setFullS
creen);
            fullScreen.addEventListener(MouseEvent.MOUSE_DOWN, removeFullS
creen, false, 0, true);
            fullScreen.gotoAndStop(3);
}else{
            fullScreen.removeEventListener(MouseEvent.MOUSE_DOWN, remove
FullScreen);
            fullScreen.addEventListener(MouseEvent.MOUSE_DOWN, setFullScre
en, false, 0, true);
            fullScreen.gotoAndStop(1);
            trace("salio de full");
     }
}
```

Lo que hacemos con este código es, en base al estado, definir los listeners para el clip **fullScreen** y mover su cabezal de frame para cambiar la apariencia del clip. Al encontrarnos en modo pantalla completa, cambia el icono del clip **fullScreen** y, al hacer clic sobre él, el usuario podrá salir de este modo y volver a la pantalla normal. Al presionar el clip **fullScreen** en pantalla completa ya no llamaremos a la función **setFullScreen** porque eliminamos su listener. En cambio, llamaremos a la función **removeFullScreen**:

```
fullScreen.removeEventListener(MouseEvent.MOUSE_DOWN, setFullScreen);
fullScreen.addEventListener(MouseEvent.MOUSE_DOWN, removeFullScreen, false,
```

```
0, true);
fullScreen.gotoAndStop(3);
```

Al ingresar a la función **removeFullScreen()**, retornamos al modo normal de la manera que podemos ver a continuación:

```
private function removeFullScreen(event:MouseEvent):void{
     stage.displayState = StageDisplayState.NORMAL;
}
```

Propiedad fullScreenSourceRect

Utilizando la propiedad **fullScreenSourceRect** del **Stage**, podemos establecer que el modo pantalla completa se aplique sobre una única parte del escenario. Esa área la definimos por medio de un rectángulo:

```
var screenRectangle:Rectangle = new Rectangle(posicionX, posicionY, ancho,
alto);
stage.fullScreenSourceRect = screenRectangle;
```

A lo largo de este capítulo hicimos un recorrido por las características más interesantes y útiles de la reproducción de videos en Flash. En los últimos tiempos, los videos se han consolidado como un elemento multimedial determinante y uno de los de mayor peso en la Web. Flash nos propone un entorno por medio del cual podemos potenciar su uso y nos permite sacar provecho de la importancia que ha adquirido el video, desde la aparición del formato FLV.

... RESUMEN

La intención de este capítulo fue brindar los conceptos necesarios para hacer un uso eficiente del video en Flash, tanto de manera interna como externa, y conocer las características y las ventajas que nos proporciona el formato FLV. En el próximo capítulo nos adentraremos en un tema que, si bien guarda relación con los conceptos hasta aquí vistos, es de una mayor complejidad: la grabación de videos desde Flash, implementando Flash Media Server 3.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1) ¿Qué es el componente **FLVPlayback**? ¿Para qué sirve? ¿Cuáles son sus ventajas y sus desventajas?

2) ¿Qué formatos de video acepta Flash?

3) ¿Cuál es la ventaja de usar videos externos?

4) ¿Qué hace la clase **NetConnection**? ¿Y la clase **NetStream**?

5) ¿Para qué son los métodos **togglePause()** y **play()** de la clase **NetStream**?

6) ¿Qué información nos proporcionan las propiedades **bytesLoaded**, **bytesTotal**, **time, bufferTime** y **bufferLenght** de la clase **NetStream**?

7) ¿Para qué utilizamos la propiedad **seek**?

8) ¿Cómo podemos averiguar la duración total de un video?

9) ¿A qué tipo de información podemos acceder por medio del evento **NetStatusEvent**?

10) ¿De qué manera hacemos que un reproductor sea fullscreen? ¿Qué evento nos informa respecto al estado en el cual se encuentra la película? ¿Cómo volvemos al modo normal?

EJERCICIOS PRÁCTICOS

1) Cree su propio reproductor de videos.

2) Asigne los controles necesarios para detener y reanudar la reproducción y algún modo que le facilite al usuario la posibilidad de modificar el volumen.

3) Genere un campo de texto que indique el tiempo transcurrido.

4) Agregue un campo de texto dinámico que informe el tiempo total.

5) Haga que su reproductor soporte el modo pantalla completa.

Cámara web, micrófono y Flash Media Server

Las nuevas tendencias y tecnologías hacen que sea necesario incluir webcams y micrófonos en nuestros proyectos de Flash. Dedicaremos la primera parte de este capítulo a los conceptos básicos para el manejo de los dispositivos y destinaremos la segunda parte para generar una aplicación integrada con Flash Media Server a fin de grabar videos.

SERVICIO DE ATENCIÓN AL LECTOR: usershop@redusers.com

INTRODUCCIÓN A WEBCAMS Y MICRÓFONOS

Las cámaras web y los micrófonos tienen, actualmente, un protagonismo que tiempo atrás no hubiésemos imaginado dentro de la plataforma de Flash, y el lenguaje ActionScript 3.0 es, en gran medida, el responsable de que esto suceda. Por medio de este lenguaje y de los avances en la transferencia de datos entre el cliente y el servidor, hoy podemos obtener resultados muy interesantes integrando micrófonos y cámaras web en nuestras aplicaciones.
En esta primera parte del capítulo sentaremos los aspectos básicos que hacen al funcionamiento y la implementación del hardware, y más adelante integraremos una aplicación con Flash Media Server 3 para grabar videos desde Flash.

Micrófono, conceptos básicos

Por medio de la clase **Microphone** podemos capturar audio desde un micrófono que esté conectado a nuestro equipo y reproducir la entrada de sonido por los parlantes, o bien enviar el audio a un servidor remoto a fin de grabarlo. Para acceder a un micrófono, lo hacemos del siguiente modo:

```
mic = Microphone.getMicrophone();
```

Es importante destacar que la clase **Microphone** no tiene un constructor como otras clases (**new Microphone**). Para tomar un micrófono, lo hacemos directamente con el método estático **Microphone.getMicrophone()**.
Una vez que se llama al método **getMicrophone()**, la aplicación nos mostrará un cuadro de diálogo con la **configuración** de Flash Player. Veremos esta ventana siempre que queramos hacer uso de micrófonos o cámaras web. Desde ella, el usuario acepta o deniega el acceso a los dispositivos de entrada. Aunque quizás pensamos que esta consulta al usuario es innecesaria, en realidad es correcto que suceda de esta manera; imaginemos lo terrible que sería si las aplicaciones pudieran acceder a nuestra webcam sin nuestra autorización.

USUARIO SIN MICRÓFONO INSTALADO EN EL SISTEMA

Es muy probable que el usuario no cuente con ningún dispositivo de entrada de audio conectado a su sistema. Para comprobar la ausencia del micrófono, podemos utilizar la propiedad **Microphone.names**, que devolverá **0**, o el método **Microphone.getMicrophone()**, que responderá **null** en caso de que no haya un dispositivo instalado.

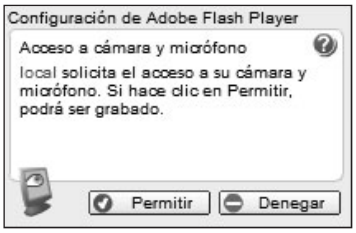


Figura 1. *Ventana de configuración de* ***Flash Player****. Desde ella se* ***acepta*** *o* ***deniega*** *el acceso a los dispositivos de cámara o a los micrófonos.*

Luego veremos algunas de las propiedades de la clase **Microphone**, pero lo que nos interesa por ahora es saber si el usuario acepta o deniega el acceso a su micrófono.

```
mic.addEventListener(StatusEvent.STATUS, onStatus, false, 0, true);
```

Por medio del evento **StatusEvent** podemos saber qué decisión tomó el usuario:

```
private function onStatus(event:StatusEvent):void{
     if (event.code == "Microphone.Unmuted"){

            trace("se permitio el acceso al mic");
            micLevel = new MicLevel();
            micLevel.x = 50;
            micLevel.y = 200;
            addChild(micLevel);
            addEventListener(Event.ENTER_FRAME, showMic, false, 0, true);

     }else if (event.code == "Microphone.Muted"){
            infoTxt.text = "no podemos hacer nada sin tu mic :(";
     }
}
```

III SISTEMAS CON MÁS DE UN MICRÓFONO

Por medio de la propiedad **names** de la clase **Microphone**, accedemos a todos los dispositivos de entrada de audio disponibles en el sistema. Para asignar un micrófono, debemos indicar su posición dentro del **array** devuelto por la propiedad **names**; por ejemplo, **Microphone.getMicrophone(2)**. De no asignar ningún valor, Flash tomará el primer dispositivo de la lista.

La propiedad **event.code** es la que nos proporciona esta información. Si su valor es **"Microphone.Unmuted"**, el usuario ha permitido el acceso a su micrófono, y en caso de que el valor sea **"Microphone Muted"**, lo ha denegado.

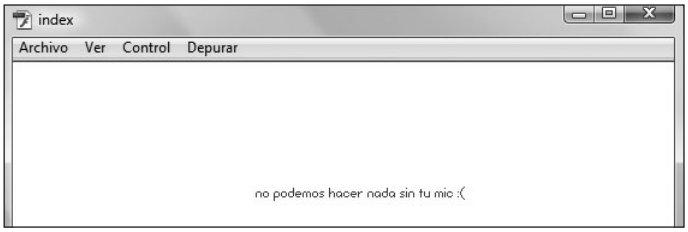


Figura 2. *Cuando el usuario deniega el acceso al micrófono, definitivamente no podemos continuar.*

Si el usuario acepta, accedemos al código que nos interesa:

```
micLevel = new MicLevel();
micLevel.x = 50;
micLevel.y = 200;
addChild(micLevel);
addEventListener(Event.ENTER_FRAME, showMic, false, 0, true);
```

Añadimos al escenario el clip **micLevel** que se encuentra en la biblioteca y agregamos un evento **ENTER_FRAME**. No podemos acceder a la información de entrada de audio por medio del método **computeSpectrum()** de la clase **SoundMixer**, pero sí contamos con una propiedad que nos informa respecto a la cantidad de sonido que detecta el micrófono por medio de la propiedad **activityLevel**. El valor de esta propiedad es un número que va de **0** (cuando no se detecta ningún sonido) a **100** (cuando se percibe un sonido alto). Utilizando esta propiedad, podemos generar una sencilla visualización, escalando un clip acorde a la **intensidad del sonido**.

```
public function showMic(event:Event){
     micLevel.back.scaleY = mic.activityLevel * 0.1;
}
```

Simplemente, aplicamos el valor de la propiedad **activityLevel** a la escala del clip nombrado **back**, que se encuentra dentro del clip **micLevel**. Recordemos que en ActionScript 3.0, los valores para las propiedades **scaleX** y **scaleY** van de **0** a **1** y no de **0** a **100**. Por este motivo, al número que obtenemos por medio de la propiedad **activityLevel** lo multiplicamos por **0.1** para obtener un valor entre **0** y **1**.

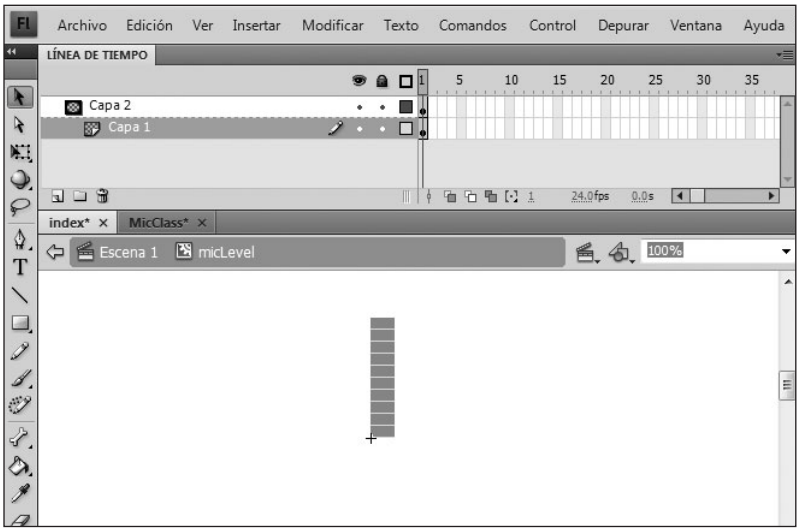


Figura 3. *Estructura del clip* ***micLevel****; el* ***registro*** *se encuentra en el borde inferior izquierdo del clip; de este modo, al escalar en* ***y*** *en un valor positivo, lo hará hacia arriba y no hacia abajo.*

Reproducción del sonido

Para reproducir el sonido capturado por el micrófono a través de los parlantes u otro dispositivo de salida instalado en el sistema usamos el método **setLoopBack()**.

```
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

Modificar parámetros del audio

A continuación veremos que podemos aplicar algunas **transformaciones** sobre el sonido antes de reproducirlo. Esto es ventajoso ya que nos permite elevar notoriamente el rendimiento del audio y obtener mejores resultados. Incluso, el buen manejo de estos parámetros es imprescindible al desarrollar aplicaciones inte-

EVITAR ACOPLES

Al reproducir por los parlantes un sonido que ingresó desde el micrófono, a veces hay un acople de audio que podría dañar el hardware. Asignando el valor **true** al método **setUseEchoSuppression()**, si bien no se elimina por completo esta posibilidad, se reduce considerablemente. **Adobe** recomienda usar este método **setUseEchoSuppression(true)** antes de llamar al método **setLoopBack()**.

grando Flash con un servidor remoto, donde el peso y la calidad de los datos enviados son de vital importancia debido a la necesidad de transferirlos.

Ganancia del micrófono: propiedad gain

Por medio de la propiedad **gain** de la clase **Microphone**, podemos definir un valor numérico entre **0** y **100** para la ganancia del micrófono, que es el valor por el cual se multiplica la señal antes de ser transmitida. Su valor por defecto es **50**.

```
mic.gain = 20;
```

Frecuencia del sonido: propiedad rate

A través de la propiedad **rate** podemos definir la frecuencia a la que el micrófono realizará la captura del sonido. Este valor se expresa en **kHz**, y las frecuencias permitidas son aquellas que admita el sistema dentro del cual se capture el audio. Pueden ser **5**, **8**, **11**, **22** o **44**, siendo **8** el valor por defecto, siempre y cuando el dispositivo de captura lo acepte. En caso contrario, el valor predeterminado será el siguiente nivel de captura por encima de los 8 kHz que admita el dispositivo que se está utilizando.

Leer y modificar el nivel de actividad

La propiedad de sólo lectura **silenceLevel** nos informa respecto a la cantidad necesaria que ha sido definida para activar el sonido. Su valor por defecto es **10**. En caso de querer modificarlo, debemos emplear el método **setSilenceLevel()**.

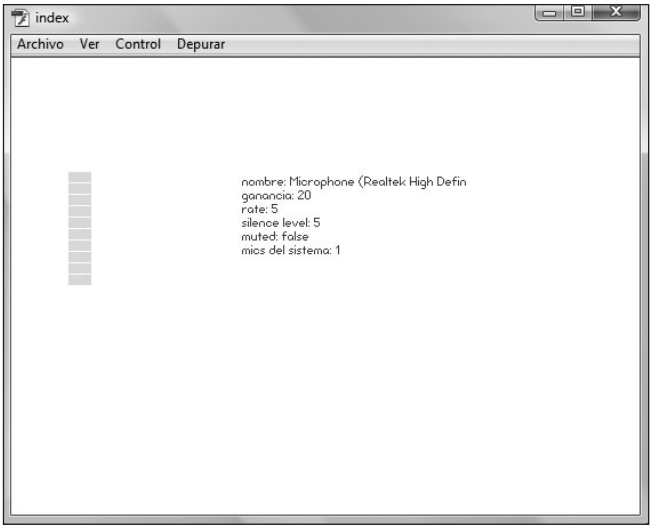


Figura 4. *Nuestra película en ejecución. En el campo de texto, algunas de las propiedades de la clase* ***Microphone****.*

A continuación observaremos otro pequeño ejemplo de cómo leer el nivel de actividad y generar una representación visual de él. Los contenidos no difieren de los que vimos hasta aquí; la única diferencia es que, en lugar de escalar un único clip en función del valor proporcionado por la propiedad **activityLevel**, utilizamos la propiedad junto a la API de dibujo de Flash para hacer una representación en el tiempo de actividad que ha tenido el micrófono.

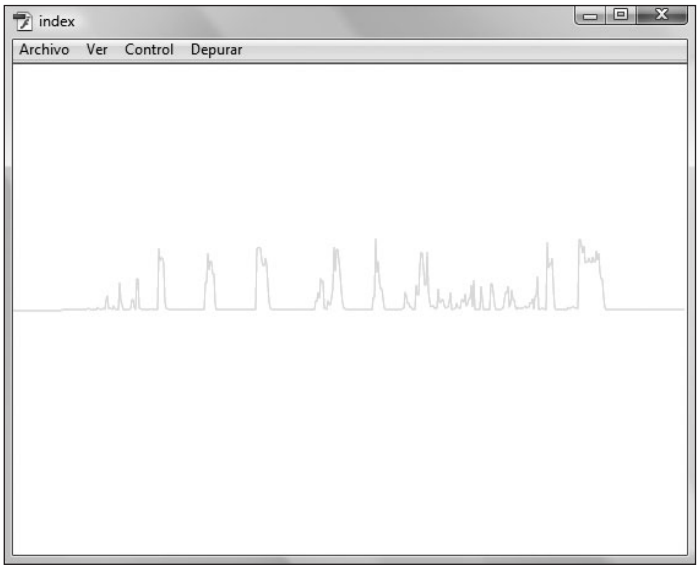


Figura 5. *Actividad del micrófono durante un período de tiempo.*

Introducción a la webcam en Flash

A diferencia del audio, donde el margen de acción es considerablemente acotado, el uso de webcams presenta una gama mucho más amplia de posibles soluciones en el desarrollo de aplicaciones. Si bien en la segunda parte del capítulo utilizaremos la cámara web para grabar video, su uso nos propone otras alternativas: desde emplearla para tomar fotos y luego experimentar libremente (dibujando sobre ellas, enviándolas al servidor para almacenarlas u otras opciones) hasta generar cap-

CORRECTO MANEJO DE LA CLASE MICROPHONE

Si bien hasta aquí el uso que estamos haciendo es más bien experimental, luego abarcaremos conceptos más avanzados, donde se requiere de una óptima configuración del micrófono a fin de sacar el mayor provecho posible al publicar audio. Conocer en profundidad los métodos y propiedades de la clase nos va a permitir una mejor configuración a la hora de exportar audio desde Flash.

turas de movimiento. Otra ventaja es que su uso puede ser tan sencillo o tan complejo como dispongamos al momento de desarrollar; lo importante es saber que, con unas pocas líneas de código, basta para tener el contenido de una webcam en pantalla. A partir de ahí, cada desarrollador puede optar por las más diversas posibilidades. Veremos que reproducir el contenido de una cámara web en Flash es mucho más sencillo de lo que seguramente pensamos:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Estas cuatro líneas de código es todo lo que necesitamos. A partir de aquí, depende de cada desarrollador hasta qué extremo llevar esta entrada de la cámara. A continuación, veremos varios **métodos** y **propiedades** de la clase **Camera**, que es la que nos permite mostrar el flujo de nuestra cámara web.
En primer lugar, seguramente notemos que existen varias similitudes con el manejo de entrada de audio. Por un lado, no tenemos un método constructor. Asignamos la cámara por medio del método **Camera.getCamera()**. Al igual que con el audio, Flash no nos permite acceder a la cámara web de un usuario sin su autorización, por lo que nos mostrará nuevamente una ventana de diálogo de Flash Player, a través de la cual debemos definir si aceptamos o denegamos el acceso de la aplicación a la webcam. Por otro lado, existe la posibilidad de que al momento de ejecutar la película, el usuario no cuente con ninguna cámara instalada. Podemos averiguar esto fácilmente del siguiente modo:

```
var cam:Camera = Camera.getCamera();
if (cam == null){
//si entramos aquí es porque el usuario no cuenta con una cámara activada.
}
```

PROPIEDAD NAME Y PROPIEDAD NAMES

No debemos confundir estas dos propiedades; por medio de la propiedad **name** conocemos el nombre del dispositivo sobre el cual establecimos un flujo de datos (la cámara que se está utilizando), y a través de la propiedad **names** obtenemos el listado de todas las cámaras disponibles con las que cuenta el sistema operativo.

Aunque es infrecuente, también puede suceder que el usuario posea más de una cámara. Para acceder al listado de dispositivos, podemos utilizar la propiedad **names** de la clase **Camera**. Al asociarla a un **array**, tendremos dentro de éste todos los dispositivos de cámaras instalados en el sistema operativo.
Como vimos en nuestro primer ejemplo, no podemos agregar una cámara al escenario. Para hacerlo, debemos vincularla a un objeto de **Video**:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Para ello, simplemente declaramos la instancia de la clase **Video** y, por medio del método **attachCamera()**, vinculamos la cámara que estamos tomando. Cuando utilizamos el método **addChild()**, lo hacemos indicando como parámetro el nombre de instancia del video y no de la cámara.

***Figura 6.** Por medio del método addChild() agregamos al escenario la instancia de video que contiene la webcam.*

III ¿QUÉ CÁMARA SELECCIONAR?

Al utilizar el método **getCamera()** de la clase **Camera** y no definir ningún parámetro, Flash asignará por defecto la primera cámara (o la única en caso de haber una sola) del listado de cámaras disponibles en el equipo. Si hay más de una, podemos acceder a ellas indicando su índice, por ejemplo, **getCámera(1)**. Lo recomendable es dejar la cámara por defecto sin ningún parámetro.

Dimensiones de cámaras y videos

Tanto en las cámaras web como en los objetos **Video** con los que trabajamos, podemos modificar sus dimensiones (alto y ancho). Existen distintos formatos y diferentes valores por defecto. Es importante que conozcamos acerca de esto a fin de obtener los mejores resultados acordes al desarrollo que debamos encarar. Lo más importante es saber que trabajamos con dos conceptos distintos: las dimensiones de una cámara web son una cosa y las del video que contendrá la cámara son otra: para modificar las dimensiones de una cámara web, debemos hacerlo por medio del método **setMode()**, indicándole el ancho (**width**), el alto (**height**) y la velocidad de fotogramas por segundo por medio del tercer parámetro, **FPS**. De no utilizar el método **setMode()**, Flash aplicará los siguientes valores por defecto:

Width: 160
Height: 120
FPS: 15

Ya vimos que hay dos modos de modificar las dimensiones de una instancia de video; al crear una nueva instancia de la clase **Video**, asignamos los parámetros **width** y **height**:

```
var miVideo:Video = new Video(320, 240);
```

O bien por medio de las propiedades **width** y **height**. Los valores por defecto para el ancho y el largo son 320 x 240.

```
var miVideo:Video = new Video();
miVideo.width = 320;
miVideo.height = 240;
```

Las dimensiones por defecto son distintas para una instancia de la clase **Video** y para el flujo de datos de una webcam, como podemos ver en la siguiente imagen.

AJUSTE DE DIMENSIONES DE MODO DE CAPTURA

Al establecer dimensiones del modo de captura, nuestros parámetros podrían no coincidir con ningún modo nativo de la cámara. En ese caso, Flash usará el modo nativo que mejor se adapte a las especificaciones. Por ejemplo, si definimos la webcam en **400, 400** y la cámara soporta 320 x 288, Flash establecerá el ancho y el largo en 288, a fin de mantener la proporción 1:1 solicitada.

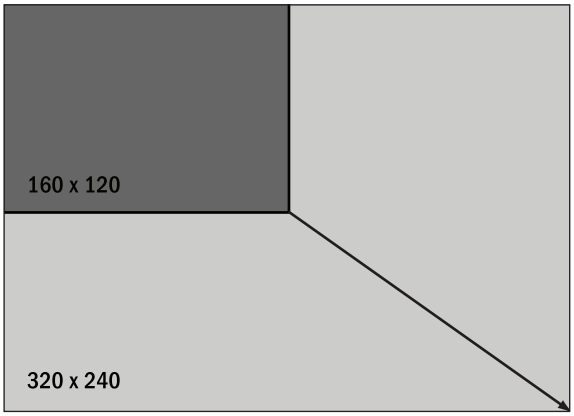


***Figura 7.** Dimensiones del flujo de datos de la webcam y de la instancia de video. El flujo de datos de una webcam se adapta al canvas de una instancia de video.*

Lo que hace Flash es adaptar las dimensiones del flujo de dato de la webcam a las dimensiones de la instancia de la clase **Video** que hayamos creado. Si prestamos atención, los valores por defecto de la webcam son la mitad, tanto en ancho como en largo, de las dimensiones por defecto de una instancia de video. Entre los formatos más comunes de las cámaras web, podemos encontrar los siguientes: 160 x 120, 176 x 144, 320 x 240, 360 x 240, 352 x 288 y 640 x 480. Por defecto, Flash utiliza la resolución 160 x 120.

La importancia de conocer las distintas dimensiones y formatos radica en que podemos seleccionar los parámetros propicios para lo que queremos lograr. Lógicamente, no es lo mismo un desarrollo experimental que requiera únicamente contar con la webcam en escena que un trabajo que implique el envío de videos a un servidor remoto, donde hay un consumo de ancho de banda y sabemos que a mayor tamaño, se utilizará mayor cantidad de recursos.

Por ejemplo, establecer las mismas dimensiones para la webcam que para el video nos dará una mejor calidad de imagen. Si bien esto es beneficioso, puede no ser lo indicado si lo que necesitamos es enviar información al servidor. Del mismo modo, utilizar un video más grande que las dimensiones de la webcam generará una peor resolución, pero a su vez, un menor tamaño.

III PROPIEDAD ACTIVITYLEVEL EN CÁMARAS

Así como hemos visto que podemos usar la propiedad **activityLevel** para conocer el grado de actividad de un micrófono, también podemos utilizarla para las cámaras web a fin de saber la cantidad de movimiento detectado. Al igual que para los micrófonos, el rango de valores va del **0** (cuando no hay movimiento) al **100** (cuando hay mucho movimiento).

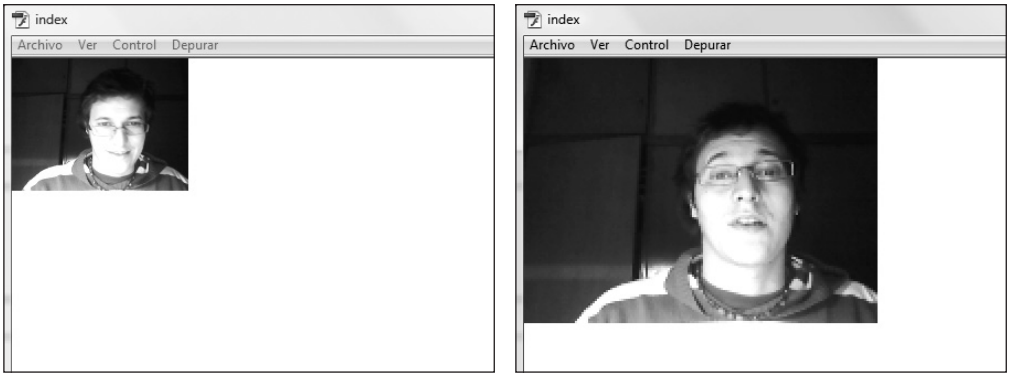


Figura 8. *Dos alternativas: a la izquierda, tanto la webcam como la instancia de video están a 160 x 120; a la derecha, la webcam está a 160 x 120 y el video a 320 x 240. Es importante notar la pérdida de calidad al escalar los contenidos.*

Desarrollo experimental con cámara web

El siguiente desarrollo es, simplemente, un ejemplo de una aplicación sencilla que podemos llevar a cabo tomando el flujo de datos de una cámara web. La mayoría de los conceptos que hacen al desarrollo ya la hemos visto en otros capítulos, por lo que priorizaremos lo referente a la captura de flujo de datos de una cámara web y cómo tomar una imagen de ella.

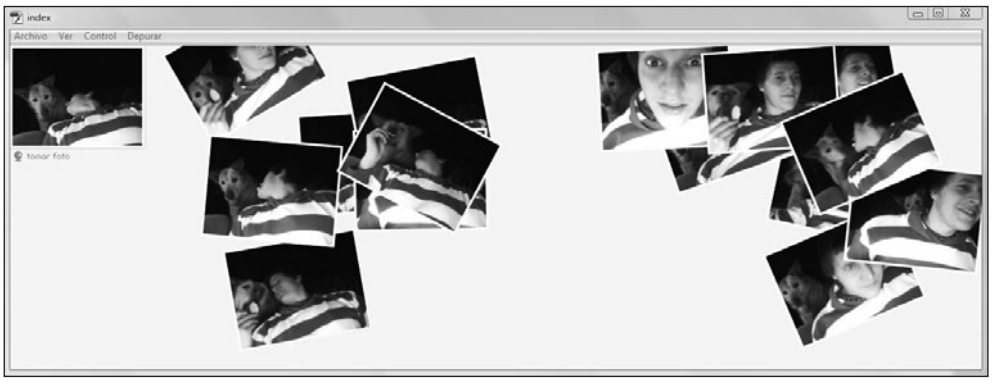


Figura 9. *Desarrollo experimental con cámara web y clase Bitmap.*

III PROPIEDAD MOTIONLEVEL Y MÉTODO SETMOTIONLEVEL()

La propiedad **motionLevel** es de sólo lectura e indica la cantidad necesaria de movimiento para considerar que hay actividad en una cámara web. Su valor predeterminado es **50** y acepta desde el **0** al **100**. Para poder modificar este valor, debemos utilizar el método **setMotionLevel()**.

A continuación, aprenderemos de qué manera podemos utilizar la cámara web para tomar imágenes. Obviaremos los eventos necesarios para verificar que el usuario cuente con una webcam, ya que puntualmente nos interesa saber de qué manera podemos tomar una imagen de una cámara web:

```
vid = new Video(160, 120);
vid.x = vid.y = 3;
container.addChild(vid);

cam = Camera.getCamera();
vid.attachCamera(cam);
```

Creamos una instancia de video y le asignamos su instancia de cámara por medio del método **attachCamera()**. Podemos ver que, al crear la instancia de la clase **Video**, asignamos a los parámetros ancho y largo las dimensiones de la cámara web, en este caso 160 x 120 pixeles. También le agregamos un marco al contenedor (**container**) y, por último, un clip que se encargará de disparar el evento que tomará una imagen de nuestra webcam:

```
webcamIcon = new WebcamIcon();
webcamIcon.buttonMode = true;
webcamIcon.mouseChildren = false;
webcamIcon.addEventListener(MouseEvent.MOUSE_DOWN, tomarFoto, false, 0,
true);
webcamIcon.x = 5;
webcamIcon.y = 130;

container.addChild(webcamIcon);

addChild(container);
```

FPS

A la hora de trabajar con aplicaciones que envían información al servidor, los FPS son un factor determinante y debemos considerarlos como un multiplicador: a mayor cantidad de fotogramas, mayor cantidad de información que se envía al servidor. Al tamaño de la imagen, debemos multiplicarlo por la cantidad de FPS que hayamos definido.

Figura 10. *Desde la captura de cámara web tomaremos una imagen y la utilizaremos de modo experimental.*

La función **tomarFoto** es la que nos interesa y la que contiene el código necesario para que podamos generar un mapa de bits con la información de la webcam:

```
private function tomarFoto(event:MouseEvent):void{

     foto = new Sprite();
     foto.mouseChildren = false;
     foto.buttonMode = true;
     foto.doubleClickEnabled = true;
     foto.addEventListener(MouseEvent.MOUSE_DOWN, startDragging, false, 0,
true);
     foto.addEventListener(MouseEvent.MOUSE_UP, stopDragging, false, 0,
true);

     miBitmapData = new BitmapData(166, 126, false);
     miBitmapData.draw(container);
     miBitmap = new Bitmap(miBitmapData);
     miBitmap.smoothing = true;
     foto.addChild(miBitmap);
     mainContainer.addChild(foto);

     var rotate:Number = getRandom(-30, 30);
     var posX:Number = getRandom(170, stage.stageWidth -
event.target.width);
     var posY:Number = getRandom(0, 300 - event.target.height);
     TweenLite.to(foto, 0.3, {x: posX, y:posY, rotation:rotate});
}
```

Cada vez que presionamos sobre el botón que se encarga de tomar las imágenes, creamos un **Sprite** al que llamamos **foto** y le asignamos algunas propiedades y eventos para poder arrastrar las imágenes una vez que las generamos y las disponemos en el escenario. Veamos a continuación cómo hacerlo:

```
foto = new Sprite();
foto.mouseChildren = false;
foto.buttonMode = true;
foto.doubleClickEnabled = true;
foto.addEventListener(MouseEvent.MOUSE_DOWN, startDragging, false, 0,
true);
foto.addEventListener(MouseEvent.MOUSE_UP, stopDragging, false, 0, true);
```

Si bien ya vimos el uso de las clases **Bitmap** y **BitmapData** en el **capítulo 3**, haremos un repaso de ellas a continuación:

```
miBitmapData = new BitmapData(166, 126, false);
miBitmapData.draw(container);
miBitmap = new Bitmap(miBitmapData);
miBitmap.smoothing = true;
foto.addChild(miBitmap);

mainContainer.addChild(foto);
```

Utilizamos la clase **BitmapData** para definir la matriz de píxeles que queremos usar para crear un mapa de bits, como vemos en las líneas siguientes:

```
miBitmapData = new BitmapData(166, 126, false);
miBitmapData.draw(container);
```

III APLICACIONES QUE UTILICEN LA WEBCAM Y EL MICRÓFONO

Si bien hay varios ejemplos en la Web, **YouTube** y **facebook** son dos de los sitios de jerarquía que emplean estas tecnologías para generar aplicaciones. En **www.youtube.com/my_webcam** hay una aplicación de **YouTube** para subir videos y **facebook** cuenta con una opción similar, que también nos permite tomar imágenes desde la cámara web.

Recordemos que las dimensiones de la webcam y de la instancia **video** las definimos en 160 y 120, pero a la clase **BitmapData** le asignamos 166 x 126 píxeles a fin de contar los márgenes que utilizamos para las fotos (3 píxeles de cada lado).

Figura 11. *Es importante distinguir entre las dimensiones de la captura de flujo de la cámara web (**160 x 120**) y las dimensiones del clip background (**166 x 126**).*

Luego, asignamos la matriz que generemos con la clase **BitmapData** a la instancia de la clase **Bitmap**. Para eso, ingresamos lo siguiente:

```
miBitmap = new Bitmap(miBitmapData);
miBitmap.smoothing = true;
foto.addChild(miBitmap);
```

Recordemos que, por medio de la propiedad **smoothing**, le aplicamos un suavizado a la imagen. Por último, generamos tres valores al azar para definir la posición **x** y la posición **y**, y aplicamos una pequeña **rotación** entre **-30** y **30**.

```
var rotate:Number = getRandom(-30, 30);
var posX:Number = getRandom(170, stage.stageWidth - event.target.width);
var posY:Number = getRandom(0, 300 - event.target.height);
TweenLite.to(foto, 0.3, {x: posX, y:posY, rotation:rotate});
```

Como vemos en el código anterior, estamos llamando a la función **getRandom()**, a la que le pasamos dos parámetros: un mínimo y un máximo. Esta función se encargará de devolvernos un número al azar entre el mínimo y el máximo que le indiquemos.

```
private function getRandom(min:Number, max:Number):Number{
     return Math.round((max - min) * Math.random() + min);
}
```

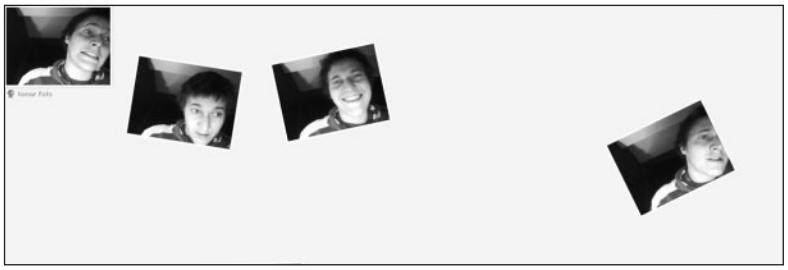


Figura 12. *Nuestra aplicación funcional: por medio de las clases* ***Bitmap*** *y* ***BitmapData*** *convertimos en mapas de bits el input de la cámara web.*

Enviar imágenes al servidor

En la segunda parte del **capítulo 3** vimos de qué manera codificar información de una imagen dentro de un **ByteArray** y enviarla al servidor. Conociendo este procedimiento, podemos utilizar la captura de la webcam de manera tal que el usuario pueda tomarse una imagen y subirla a un servidor, o bien generar su descarga. Recordemos que para este tipo de operaciones, necesitamos de un lenguaje del lado del servidor. Hemos utilizado PHP a lo largo de nuestros desarrollos y también lo requeriremos para este ejemplo en especial, por lo que para su correcto funcionamiento, debemos contar con un servidor local instalado o bien ejecutarlo en un servidor con soporte para PHP.

Figura 13. *Al hacer clic en los thumbnails, los enviaremos al servidor y los almacenaremos en carpetas.*

Este ejemplo es una extensión del anterior. La variación es que agregamos un Event Listener sobre cada imagen que generamos de la toma de la cámara web para que, al hacer doble clic, enviemos la imagen al servidor:

```
foto.doubleClickEnabled = true;
foto.addEventListener(MouseEvent.DOUBLE_CLICK, enviarFoto, false, 0, true);
```

La función **enviarFoto()** será la que se encargará de mandar la imagen codificada al archivo **PHP** que se ocupará de almacenarla. En caso de no recordar este procedimiento, en el **capítulo 3** hemos explicado de qué manera se utiliza la clase **as3CoreLib** en la función **enviarDibujo()**. Ambas funciones realizan la misma tarea, sólo que, en aquel capítulo, convertíamos en **mapa de bits** un dibujo hecho por medio de la API de dibujo de Flash y en este caso transformamos en un mapa de bits una imagen tomada de la captura de la cámara web:

```
private function enviarFoto(event:MouseEvent):void{

     var enviarFoto:Sprite = Sprite(event.target);
     var sendBitmapData:BitmapData = new BitmapData(166, 126, false);
     sendBitmapData.draw(enviarFoto);
     var byteArray:ByteArray = new JPGEncoder(90).encode(sendBitmapData);
     var urlRequest:URLRequest = new URLRequest();
     urlRequest.url = "http://localhost/imagen/php/image.php";
     urlRequest.contentType = 'multipart/form-data; boundary=' + Upload
PostHelper.getBoundary();
     urlRequest.method = URLRequestMethod.POST;
     urlRequest.data = UploadPostHelper.getPostData('file.jpg', byteArray );
     urlRequest.requestHeaders.push( new URLRequestHeader( 'Cache-Control',
'no-cache' ) );
     var urlLoader:URLLoader = new URLLoader();
     urlLoader.dataFormat = URLLoaderDataFormat.BINARY;
     urlLoader.load(urlRequest);

}
```

{ } CAPTURA DE MOVIMIENTO

Un tema verdaderamente interesante, pero que no podremos tratar debido a las prioridades del capítulo es la captura de movimiento. Es posible generar aplicaciones experimentales muy interesantes por medio de la captura de nuestros movimientos que tome la cámara web. Hay varios recursos, ejemplos y proyectos experimentales en Internet que vale la pena detenerse a mirar.

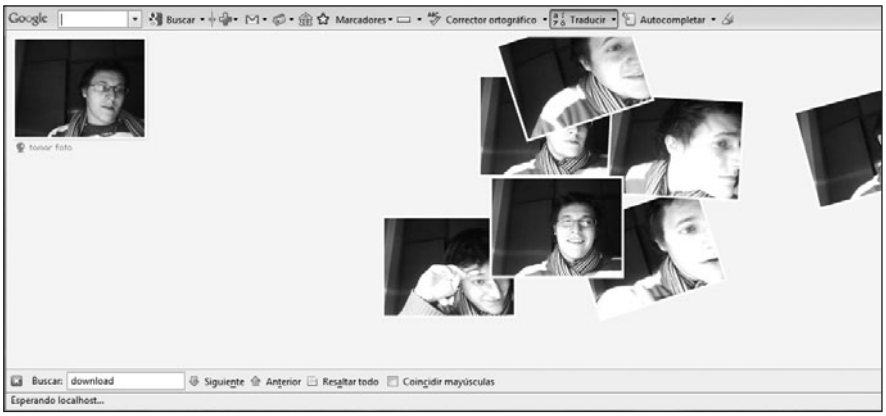


Figura 14. *La imagen se está enviando al servidor: la Barra de estado esperando por una respuesta del servidor nos informa al respecto.*

Como ya mencionamos, para efectivizar el envío de las imágenes requerimos de un lenguaje del lado del servidor. En el **capítulo 3** hemos creado un archivo al que llamamos **image.php**. Lo utilizaremos también en este capítulo ya que lo único que necesitamos es almacenar la imagen recibida desde Flash. Dentro de ese archivo, definimos como ruta para las imágenes una carpeta a la que llamaremos **imagenes**. En caso de querer cambiar el destino, debemos crear otra carpeta y modificar la ruta dentro del archivo **php/image.php**.

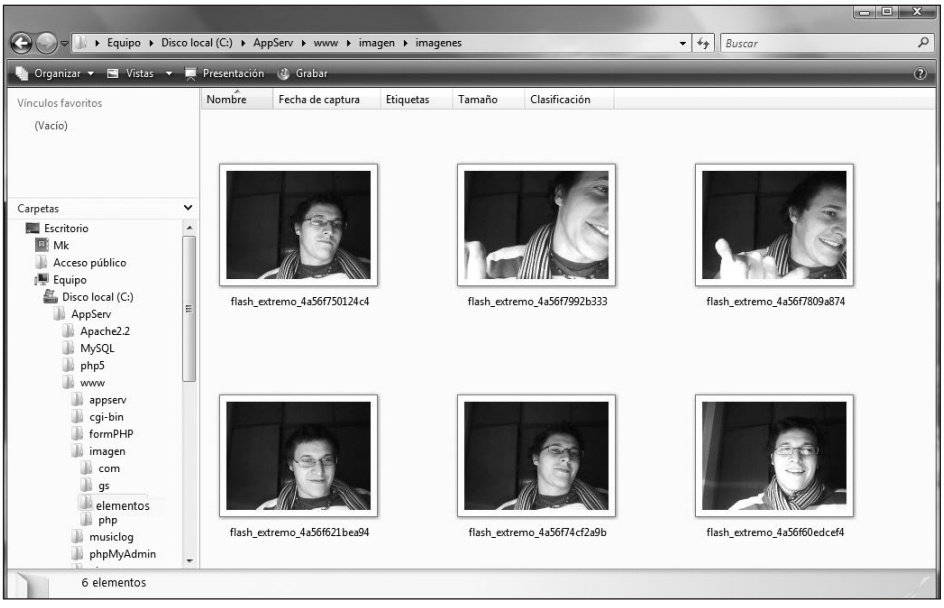


Figura 15. *Carpeta imágenes: desde Flash enviamos las imágenes a PHP, y el archivo PHP las guarda con un* ***identificador único*** *dentro de la carpeta imagenes.*

En realidad podríamos dedicar un libro entero al desarrollo de aplicaciones y a la experimentación haciendo uso de la cámara web y el micrófono, pero debemos adentrarnos en un tema nuevo y muy requerido en el mundo Flash: la **grabación de video**. Esta primera parte del capítulo deja sentadas las bases para manejar cámaras web y micrófonos con solvencia. A continuación, veremos de qué manera integrar un desarrollo hecho en Flash con **Flash Media Server 3**, para poder grabar videos desde el primero.

INTRODUCCIÓN A FLASH MEDIA SERVER 3

Hemos nombrado **Flash Media Server 3** (o **FMS3**) en reiteradas oportunidades, pero aún no hablamos en detalle de él. Es un **servidor de socket abierto**. La diferencia entre un servidor de este tipo con un servidor web es que FMS3 mantiene una conexión constante entre el servidor y el cliente (**Flash**). En un servidor web normal, se solicita información desde el cliente, el servidor la gestiona y la devuelve, y luego se cierra la conexión. En cambio, con Flash Media Server 3 se establece una conexión y ésta queda abierta de forma permanente hasta que nosotros decidamos cerrarla o se cierre la aplicación (al cerrar el explorador, por ejemplo). Mientras esto sucede, podemos enviar y recibir cualquier tipo de información en tiempo real. Que esta conexión se mantenga abierta es lo que nos permitirá grabar video o audio desde Flash. Con un servidor normal, no podríamos hacerlo. Flash se **comunica** con Flash Media Server 3 por medio de un **protocolo** denominado **RTMP**, acrónimo de **Real Time Messaging Protocol**. Luego, veremos más al respecto de éste.

Los posibles desarrollos en FMS3 son innumerables y podrían dedicarse varios libros al tema, pero no es nuestra intención adentrarnos en Flash Media Server 3, ni tampoco es nuestro objeto de estudio. La finalidad de este capítulo es que conozcamos acerca de él, que tengamos una aproximación a este tipo de servidor y que veamos de qué manera utilizarlo para grabar video y audio en tiempo real. Esto tomó una gran relevancia en los últimos tiempos, producto del desarrollo y crecimiento de la Web

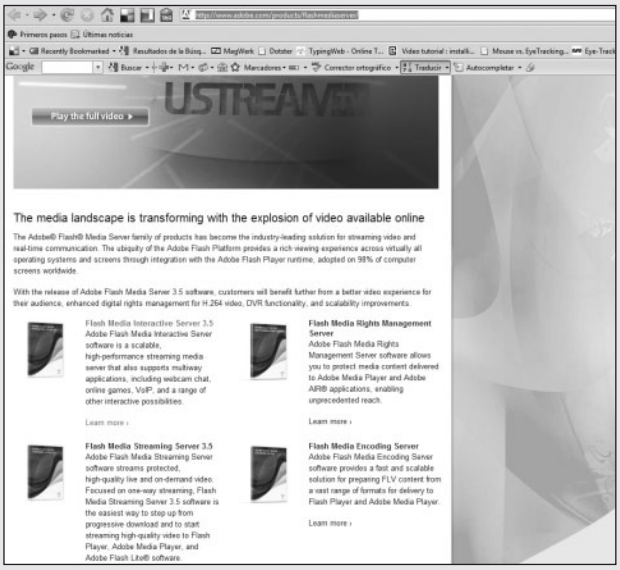
ALTERNATIVAS DE FMS3

FMS3 cuenta con tres versiones: **FMIS3** es la versión completa, **FMSS3** se limita al streaming de video y audio y **FMDS3**, versión para desarrolladores, cuenta con las mismas características que **FMIS3**, pero sólo soporta 10 conexiones y no se puede usar para desarrollos comerciales, pero sí para aplicaciones locales o LAN. **FMIS3** es gratuita y será la que emplearemos en este capítulo.

2.0 y es un tema constantemente solicitado para diversos sitios (y lo seguirá siendo durante mucho tiempo), por lo que vale la pena que le dediquemos un capítulo.
Como **requerimientos** para instalar Flash Media Server, Adobe recomienda contar con un procesador de 3.2 GHz Intel Pentium 4 y 2 Gb de memoria RAM (4 Gb recomendados). Al momento de escribir esta obra, se ha testeado su funcionamiento en Windows Vista de 32 y 64 bits y en Windows XP de 32 bits, corriendo correctamente en ambos (en una PC con procesador Intel Core 2 duo de 2.4 GHz y 4 Gb de memoria RAM). Flash Media Server 3 se encuentra disponible únicamente en inglés, y para instalarlo debemos realizar el procedimiento que veremos a continuación.

■ Cómo instalar FMS3 en un sistema — PASO A PASO

1 Ingrese al sitio **www.adobe.com/products/flashmediaserver/** y seleccione la versión de **FMS3** que desea instalar. Para el desarrollo que llevaremos a cabo, debe instalar **Flash Media Interactive Server 3.5** en su versión para desarrolladores.

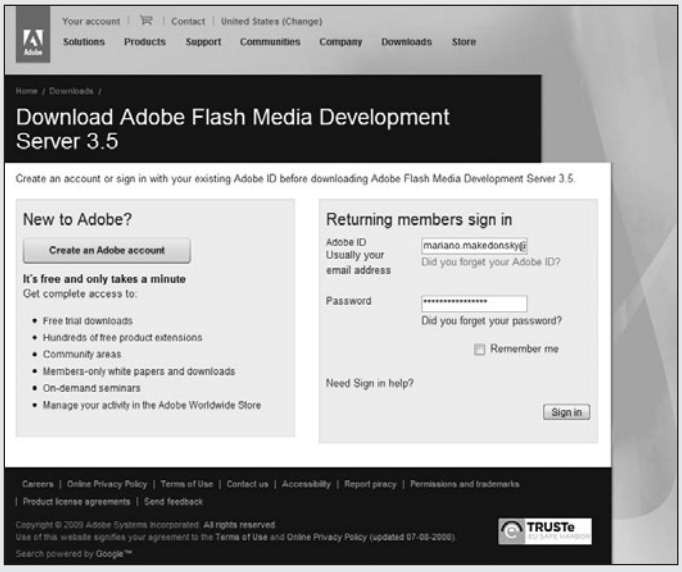


RED5, ALTERNATIVA GRATUITA A FLASH MEDIA SERVER

Si bien **Flash Media Server** es pago, contamos con una alternativa gratuita: **RED5**, un servidor **RTMP** open source que nos permite llevar a cabo muchos de los procesos que **FMS3** nos posibilita, como grabar videos, hacer streaming de contenidos, etcétera. En el sitio de **RED5** (**http://osflash.org/red5/**) encontrará información al respecto.

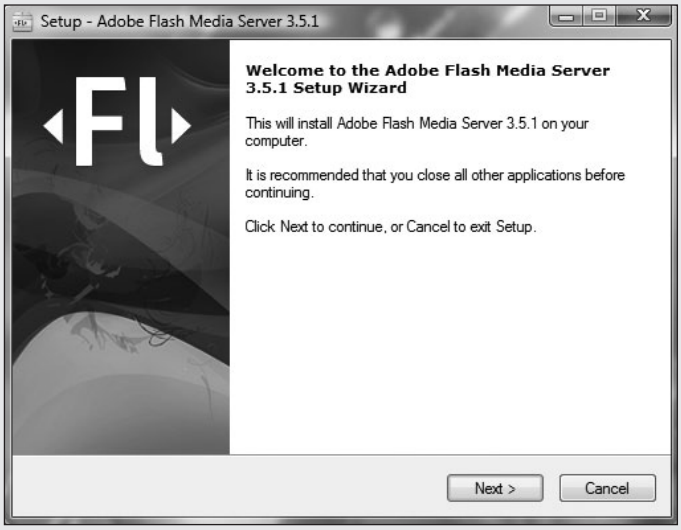
2 Cuando sea redireccionado a una nueva página, seleccione, del menú de la derecha, la opción **Download Free Adobe Flash Media Development Server 3.5**.

3 Para acceder a la descarga, debe estar registrado. Si ya cuenta con ID y password, proceda al login en el sitio. En caso contrario, regístrese antes de continuar.

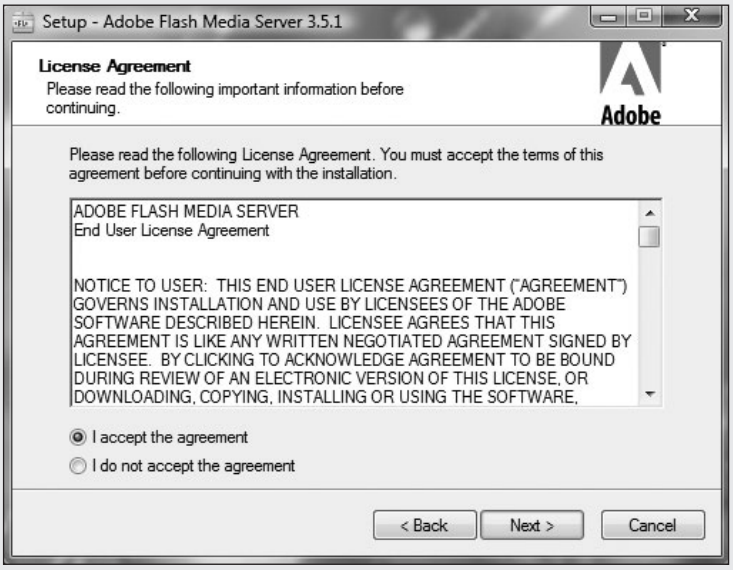


4 Seleccione el idioma y el sistema operativo en el que va a instalar **Flash Media Server 3**. Luego, presione el botón **Download** para comenzar la descarga del contenido que eligió.

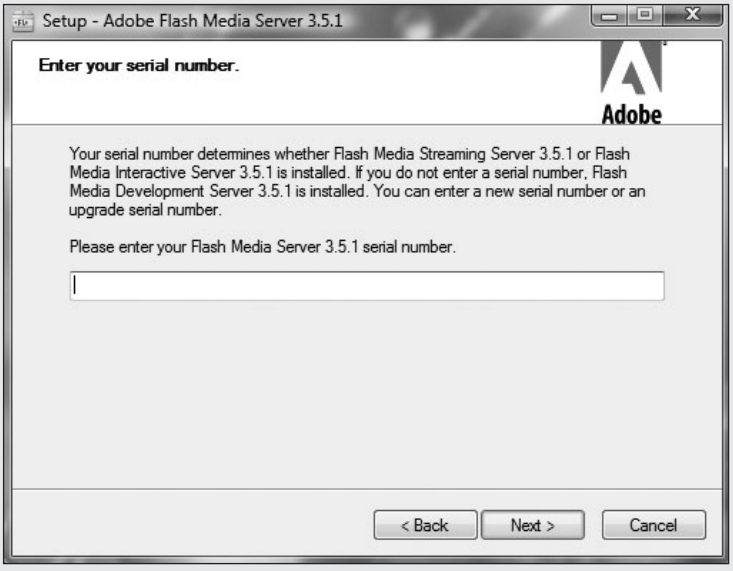
5 Una vez que finaliza la descarga, descomprima el archivo y ejecute el instalador (**FlashMediaServer3.5.exe**). Cuando inicie el proceso de instalación, presione el botón **Next** para poder comenzar con el proceso.



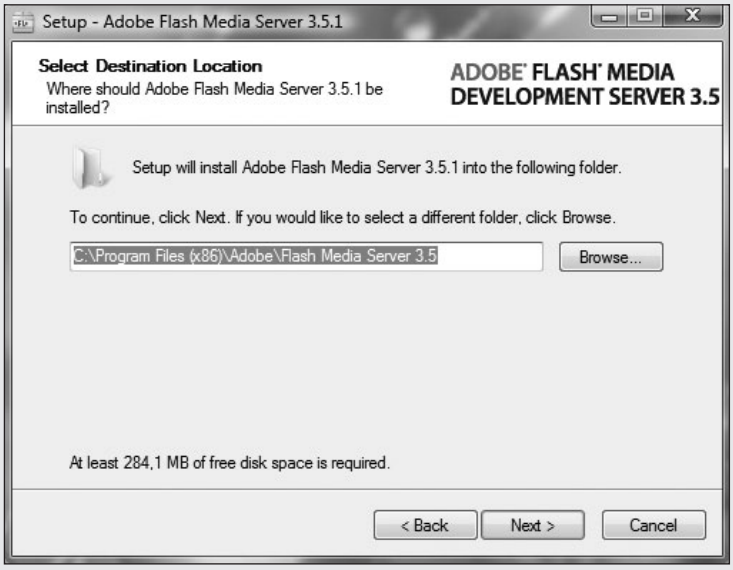
6 Lea las condiciones de uso y marque la opción **I accept the agreement** para aceptarlas y continuar. Haga clic en **Next**.



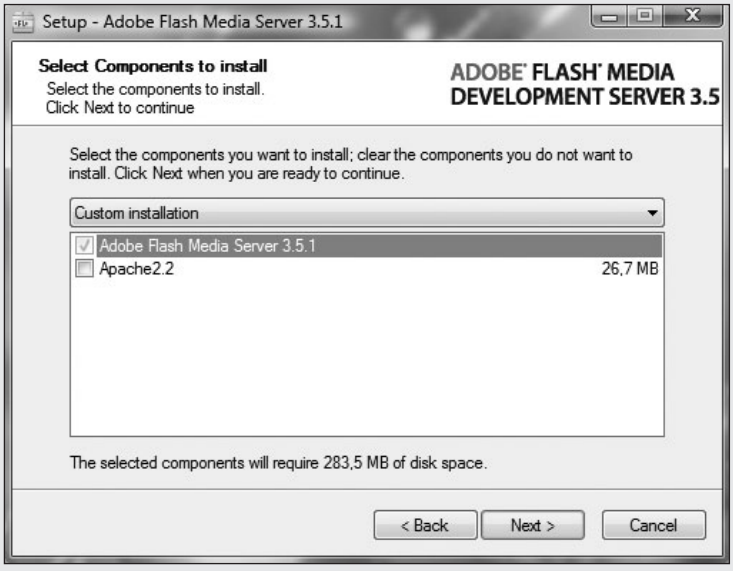
7 Esta ventana es muy importante dado que solicita el número de serie para utilizar el producto. En caso de contar con una licencia propia, utilícela aquí. En caso contrario, deje el espacio para el serial vacío y presione **Next**. De esta manera, se instalará la versión gratuita para desarrolladores.



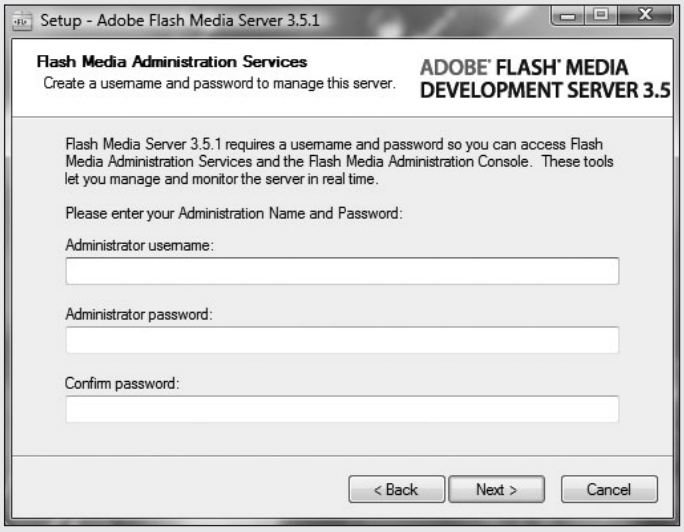
8 Luego, defina el directorio en el cual instalará el servidor.



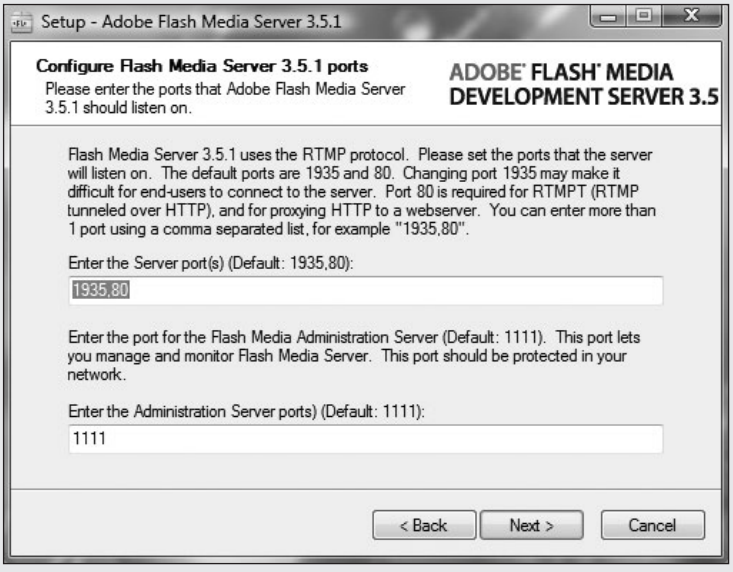
9 Una vez definido el directorio, Flash Media Server brinda la opción de instalar otro componente. En la lista **Custom Installation**, puede seleccionar la opción de instalar un servidor **Apache** junto a FMS3 (en este caso no será necesario hacerlo ya que trabajamos con PHP en otras oportunidades, por lo que deberíamos tener el servidor Apache instalado).



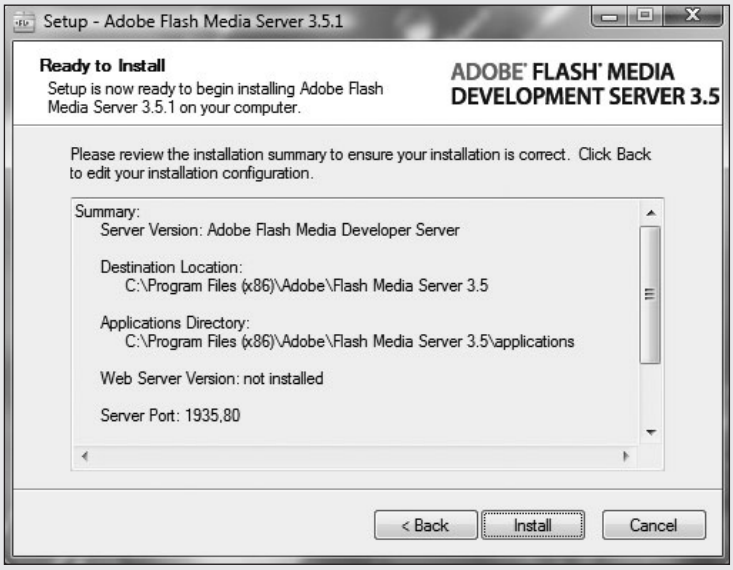
10 La siguiente ventana es fundamental ya que luego se deberá aplicar esta información al utilizar la consola de administración de FMS3. Recuerde el nombre de usuario y la contraseña que ingrese. Esta información es **case-sensitive** (distingue entre minúsculas y mayúsculas).



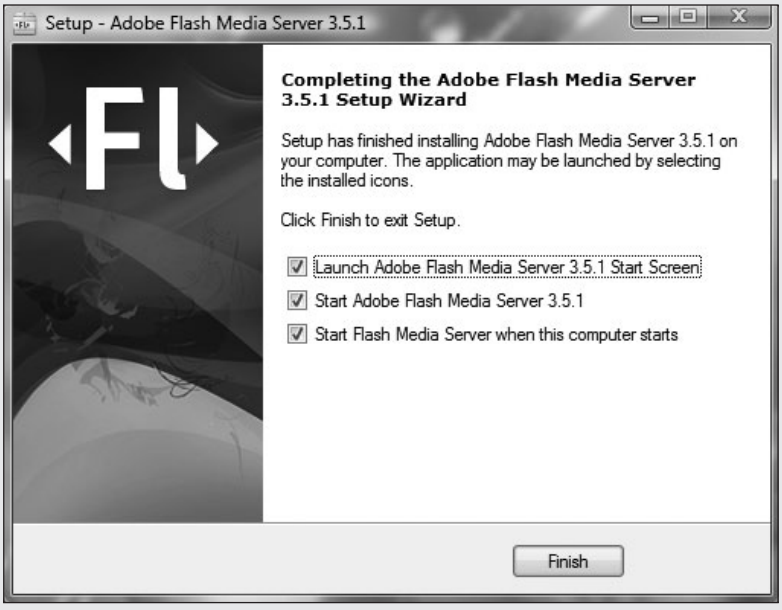
11 Defina los puertos que utilizará FMS3. A menos que tenga algún servicio empleando esos puertos, lo recomendable es dejar los valores que FMS3 provee por defecto. Presione **Next** para continuar.



12 Una vez concluida la configuración, se presentará un resumen de las opciones elegidas y, luego de presionar **Install**, comenzará la instalación de FMS3.



13 Cuando finaliza la instalación, puede iniciar el servicio y definir que éste arranque al iniciarse el sistema. Esta decisión depende de cada usuario y del uso que vaya a hacer del servidor. Pulse **Finish** para terminar.



¡Eso es todo! Con estos sencillos pasos, tenemos instalado localmente **Flash Media Server 3**. Para corroborar que el servicio está corriendo sobre nuestro sistema, podemos observar los **servicios**. Una vez que contamos con el servidor y éste se encuentra activo en nuestro sistema operativo, debemos ver de qué manera conectar nuestra película Flash con él, a fin de establecer un canal entre ambos para el envío y la recepción de la información.

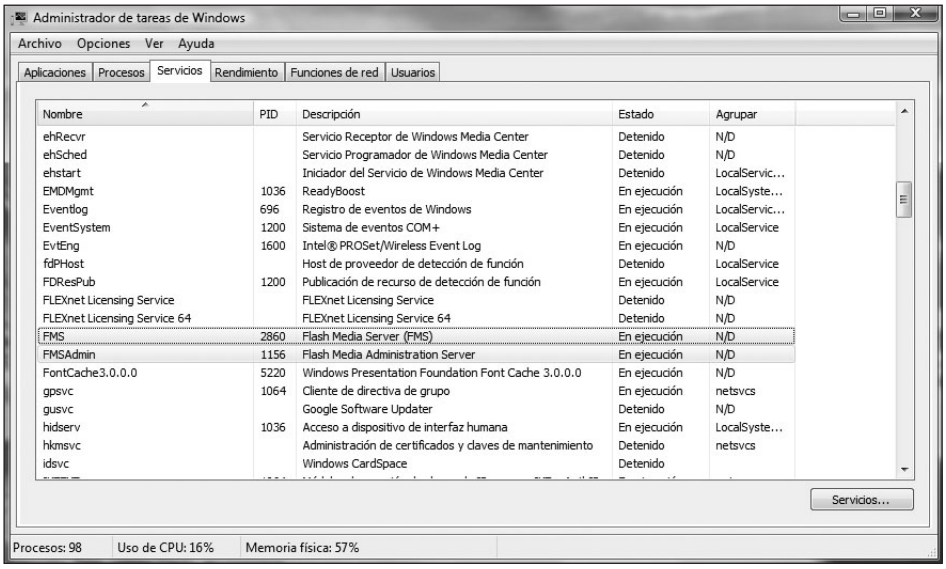


Figura 16. *Dentro del Administrador de tareas, en la pestaña* ***Servicios****, vemos* ***FMS*** *y* ***FMSAdmin****. Eso demuestra que el servidor está corriendo correctamente sobre nuestro sistema operativo.*

Preparar el entorno en Flash Media Server 3

Cuando llevamos a cabo la instalación del servidor, se generó en nuestro equipo un directorio con todos los archivos de éste. Si no cambiamos la ruta de destino durante la operación, esta carpeta se encuentra en **c://archivos de programas/adobe/Flash Media Server 3.5**. Dentro de ella, encontraremos una carpeta que se llama **applications**.

DOCUMENTACIÓN DE FMS3

Del sitio web **www.adobe.com/products/flashmediaserver/pdfs/fms3_5_wp_ue.pdf** es posible descargar un archivo PDF (únicamente en idioma inglés) con la documentación oficial de **Flash Media Server 3**. Puede ser de utilidad tanto para consultas como para profundizar en el tema.

Figura 17. *Carpeta dentro del sistema operativo donde se instalaron los directorios y subdirectorios. Dentro de ella encontraremos la carpeta* ***applications****.*

En la carpeta **applications**, debemos crear una carpeta con el nombre de la aplicación que vamos a llevar a cabo. El nombre de la carpeta deberá ser el nombre de nuestra aplicación. Tratándose de nuestro primer ejemplo práctico, donde lo que queremos es establecer una conexión entre Flash y Flash Media Server 3, llamaremos a nuestra aplicación **conexion**.

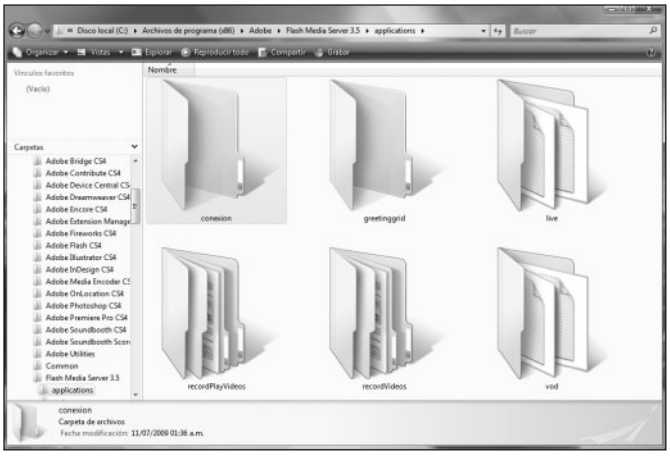


Figura 18. *Dentro del directorio* ***applications****, generamos una carpeta que llevará por nombre el de nuestra aplicación;* ***conexion*** *en este primer ejemplo.*

Conectar Flash con FMS3

Ya instalamos el servidor, se encuentra corriendo en nuestro sistema y tenemos creada nuestra aplicación (**conexion**). Lo que nos resta ver es de qué manera es-

tablecemos, desde Flash, una conexión con Flash Media Server 3. Para conectarnos, debemos utilizar la clase **NetConnection**:

```
nc = new NetConnection();
nc.addEventListener(NetStatusEvent.NET_STATUS, checkStatus, false, 0,
true);
nc.connect("rtmpe:/conexion");
```

Si bien esta clase ya la hemos visto en el capítulo anterior, hay una diferencia radical en su uso: la habíamos utilizado para cargar archivos de forma progresiva, no para establecer una conexión con un servidor. Es por esto que, en el capítulo anterior, definíamos con el valor **null** a la conexión:

```
nc.connect(null);
```

Mientras que en este caso, debemos indicarle el nombre de la aplicación a la cual queremos conectarnos, como vemos en las siguientes líneas:

```
nc.connect("rtmpe:/conexion");
```

Ahora veamos algunas consideraciones y nuevos conceptos que debemos tener en cuenta: como hemos dicho anteriormente, al utilizar el método **connect()** de la clase **NetConnection** y querer establecer una conexión con un servidor **RTMP**, debemos indicar como parámetro la ruta de la aplicación con la que necesitamos conectarnos. Si bien no es complicada, debemos tener en cuenta que esta sintaxis presenta algunas particularidades, que veremos a continuación:

- **Flash** y **FMS3** se conectan por medio del protocolo **RTMP** (**RTMPE** es su última versión, donde la E corresponde a **Encryption**).

PROTOCOLO RTMP O RTMPE

Cuando establecemos una conexión entre Flash y Flash Media Server 3, lo hacemos por medio del protocolo **RTMP**. Su última versión es **RTMPE**: Real Time Messaging Protocol Encryption. **RTMP** es más veloz, pero **RTMPE** es un protocolo más seguro. En caso de que no sea posible conectarse a través del protocolo RTMPE, conviene intentarlo a través del protocolo RTMP.

- Cuando nos conectamos a una aplicación local que se encuentra en nuestra computadora, tenemos que utilizar una única barra y no dos. Empleamos dos cuando debemos conectarnos a un servidor remoto, pero no es éste nuestro caso.
- Indicamos el nombre de la aplicación a la que nos queremos conectar. Pasos atrás la llamamos **conexion** y se encuentra en la carpeta **applications** de Flash Media Server 3.

Con **NetStatusEvent** conocemos el estado de la conexión y el mensaje devuelto:

```
nc.addEventListener(NetStatusEvent.NET_STATUS, checkStatus, false, 0, true)

private function checkStatus(event:NetStatusEvent):void {

     statusTxt = new TextField();
     statusTxt.autoSize = TextFieldAutoSize.LEFT;
     statusTxt.x = 50;
     statusTxt.y = 10;
     addChild(statusTxt);

     var status:String = event.info.code

     trace(status);

     if( status == "NetConnection.Connect.Success"){
            trace("joder...todo fue bien");
            statusTxt.text = "...todo fue bien, esta deberia ser tu primer
conexion entre Flash y FMS3 :)";
     }else if(status == "NetConnection.Connect.Failed"){
            statusTxt.text = "algo hicimos mal :(";
     }else if(status == "NetConnection.Connect.Rejected"){
            statusTxt.text = "no hay permiso para acceder a la aplicacion";
}
```

EVENTO NETSTATUSEVENT

Recordemos que en el **capítulo 7** hemos abarcado el evento **NetStatusEvent**, que es el encargado de informarnos respecto a los distintos estados en los cuales se encuentra una conexión o el streaming. En ese capítulo encontramos las distintas alternativas que nos propone el evento, tanto para la clase **NetConnection** como para la clase **NetStream**.

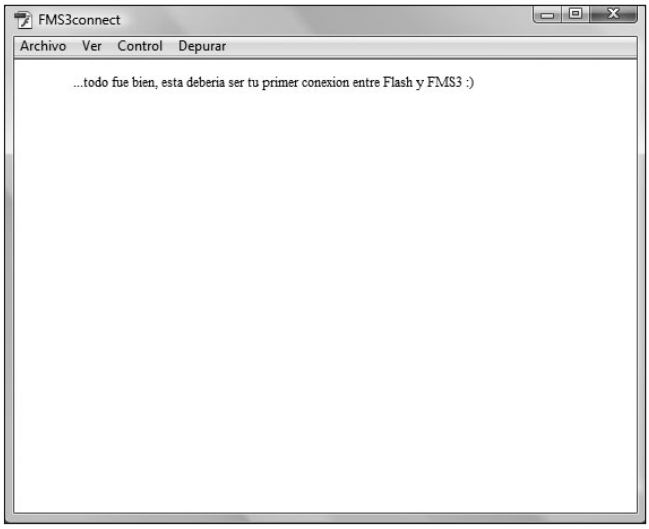


Figura 19. *Si todo va bien y hemos realizado todos los pasos que vimos hasta aquí, deberíamos obtener este mensaje de conexión exitosa.*

Si obtuvimos ese mensaje, acabamos de establecer una conexión entre Flash y Flash Media Server 3. La mejor manera de interpretar esta conexión es como un **canal** que se estableció entre ambos y que va a quedar abierto hasta que cerremos la aplicación o demos la orden de que se termine la conexión. Esta conexión a la que hacemos mención se da a través del protocolo RTMP o RTMPE.

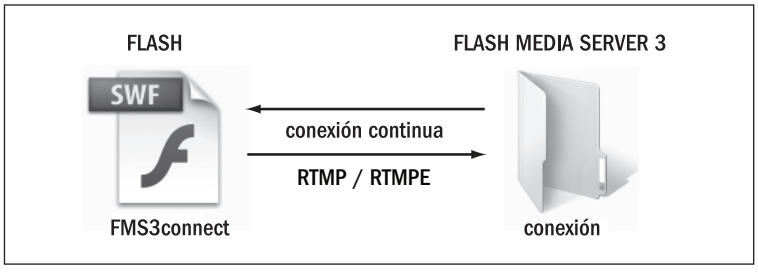


Figura 20. *Flujo de datos entre una aplicación Flash y Flash Media Server 3.*

OTRAS APLICACIONES CON SERVIDORES RTMP

Si bien el envío y la recepción de audio y video son sus principales cualidades y ventajas y a la vez los recursos más demandados, también se pueden emplear estas tecnologías para generar **videoconferencias**, **salas de chat**, **pizarras**, **juegos multiusuario** y varias otras aplicaciones que requieran de dos o más usuarios conectados en simultáneo interactuando por medio de una interfaz.

Este canal que se establece es lo que nos permite realizar los desarrollos que con un servidor web común no podríamos; por medio de esa conexión es que podemos enviar y recibir información en tiempo real. Desde el momento que creamos esa conexión, es posible dar comienzo a nuestros desarrollos.

Grabar videos

A continuación veremos de qué manera grabar videos desde Flash, utilizando nuestra cámara web como soporte para la imagen y el micrófono para el audio.

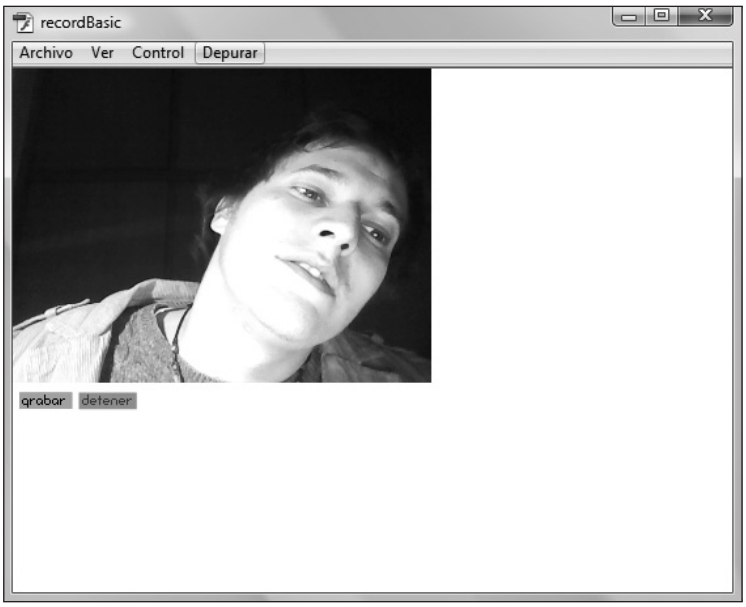


Figura 21. *Nuestra aplicación para grabar video desde Flash.*

En primer lugar, establecemos la conexión con FMS3. Llamaremos a esta aplicación **recordVideos**. Recordemos que debemos generar una carpeta con este mismo nombre dentro de la carpeta **applications** del directorio **c://archivos de programas/adobe/Flash Media Server 3.5** a fin de crear la aplicación y luego poder establecer la conexión.

```
nc = new NetConnection();
nc.addEventListener(NetStatusEvent.NET_STATUS, checkStatus, false, 0,
true);
nc.connect("rtmpe:/recordVideos");
```

Una vez establecida la conexión, llamamos a dos funciones: **addButtons()** para agregar a la interfaz los dos botones que se encargan de comenzar y detener la

grabación y **configWebcamMic()** que, como su nombre lo indica, la utilizamos para definir todos los métodos y propiedades para optimizar la cámara web y el micrófono para este proyecto, como vemos a continuación:

```
if( status == “NetConnection.Connect.Success”){
     addButtons();
     configWebcamMic();
}
```

Lo que en verdad nos interesa analizar es la función **configWebcamMic()**, ya que hay algunos conceptos interesantes para explorar y **sintaxis** que no hemos visto hasta aquí:

```
cam = Camera.getCamera();
cam.setKeyFrameInterval (20);
cam.setMode (320,240,15,false);
cam.setMotionLevel (30,3000);
cam.setQuality (40000,0);

video = new Video(cam.width, cam.height);
video.attachCamera(cam);
addChild(video);

mic = Microphone.getMicrophone();
mic.gain =85;
mic.rate=11;
mic.setSilenceLevel (30,1000);
mic.setUseEchoSuppression (true);

ns = new NetStream(nc);
ns.attachCamera(cam);

ns.attachAudio(mic);
```

En lo que respecta a la cámara web, hemos dicho en reiteradas ocasiones que los parámetros que definamos en sus propiedades y métodos tienen relación directa con el tipo de desarrollo que estamos llevando a cabo. Si es una **aplicación** que corre sobre un sistema operativo y no hace uso de una conexión de Internet para el envío y la recepción de datos, probablemente obviemos varias consideraciones (no hay anchos de banda, ni tiempos de descarga, ni distintas velocidades de conexión),

pero cuando vamos a desarrollar un proyecto para la Web, debemos hacer un balance entre varios factores. En primer lugar, no todas las conexiones son iguales, por lo que el ancho de banda cambiará dependiendo de la conexión con la que cuente cada usuario. Por otro lado, tenemos que tomar decisiones sobre qué es lo más importante en nuestro proyecto; si la **calidad** de las imágenes o la solvencia del proyecto, y esto generalmente presenta un dilema: a mayor calidad, mayor ancho de banda y mayor cantidad de recursos; a menor calidad, menos ancho de banda, pero también menos calidad de imagen. Estas variables las debemos conocer y manejar, y dependen en parte del tiempo y de la experiencia que podamos adquirir y del enfoque que hagamos de nuestros proyectos. De todas maneras, lo importante es saber que siempre vamos a tener que **priorizar** y, en estos casos, generalmente implica relegar.

Definir fotogramas clave

Por medio del método **setKeyFrameInterval()** definimos cada cuántos fotogramas del video se va a transmitir un **fotograma clave**. El mejor modo de interpretar el concepto de fotograma clave es pensando los videos como una sucesión de imágenes. Cada una de esas imágenes es un fotograma. De cada uno de esos fotogramas, por medio del método **setKeyFrameInterval()**, podemos definir un valor numérico que indica cada cuántos fotogramas habrá un fotograma clave, aquél que al enviarlo al servidor no se le aplica ningún tipo de **interpolación**. A los fotogramas que se encuentran entre dos fotogramas clave, se los denomina **fotogramas interpolados**.

```
cam.setKeyFrameInterval(20);
```

Un claro ejemplo donde se puede apreciar este concepto es cuando utilizamos el método **seek()** durante la reproducción de un video a fin de desplazarnos en el tiempo. Lo que hace este método es colocar la cabeza lectora del video en el fotograma clave más próximo que ubique. Notaremos que en algunos videos el desplazamiento es prácticamente continuo y que en otros aparecen grandes saltos. Esta diferencia se debe, justamente, a la cantidad de fotogramas clave que se han incluido en un video. A modo de conclusión, podemos establecer que:

- A mayor cantidad de fotogramas clave, mayor calidad y más fluidez en la reproducción del video, pero en caso de estar enviando los datos del video a un servidor, se hará un mayor uso de los recursos del ancho de banda.
- A menor cantidad de fotogramas clave, menores calidad, peso y fluidez en la reproducción de un video, lo que requiere de menos recursos al trabajar con un servidor remoto, donde el envío de datos y el ancho de banda son factores determinantes para el rendimiento de una aplicación.

En definitiva, no hay una técnica por medio de la que podamos definir un número ideal de fotogramas clave para un video. Esto tiene relación directa con lo que hemos mencionado anteriormente; manejar correctamente estos parámetros no implica emplear valores por defecto sino saber qué valores son óptimos para nuestro desarrollo en particular. Entonces, si desarrollamos una aplicación que reproduce videos y estos son el factor distintivo, se deberá priorizar la calidad y, por lo tanto, definir una mayor cantidad de fotogramas clave.
En cambio, si lo que estamos desarrollando es una aplicación como la nuestra en la que debemos usar recursos de ancho de banda para el envío de datos, debemos optimizar nuestro desarrollo para no consumir todo el ancho de banda del que disponemos en unos pocos videos.

Definir prioridades: calidad o ancho de banda

El método **setQuality()** es fundamental cuando desarrollamos aplicaciones para la Web. Por medio de él definimos qué es más importante para nuestra aplicación: si el uso del ancho de banda o la calidad de la imagen. En caso de que nuestra prioridad sea el ancho de banda, debemos indicar su valor y pasarle el valor **0** al segundo parámetro, como vemos a continuación:

```
cam.setQuality (40000,0);
```

De esta manera, Flash transmitirá el video con la máxima calidad posible dentro del ancho de banda que hayamos especificado. En caso de que se supere ese ancho de banda, Flash reducirá la calidad de imagen para evitar superarlo.
Si decidimos que la calidad del video tiene prioridad sobre el ancho de banda, debemos establecer **0** para el primer parámetro y utilizar el segundo para indicar la calidad de la imagen. En caso de ser necesario, Flash va a reducir la velocidad de fotogramas a fin de mantener la calidad del video.

```
cam.setQuality (0,90);
```

NOMBRE PARA LOS ARCHIVOS

Al publicarse un video, si ya existe uno con el mismo nombre, FMS3 lo reemplazará por el nuevo. Para generar nombres únicos podemos crear cadenas de caracteres, utilizando la clase **Date()** de Flash, con fechas y horas o, en caso de utilizar un lenguaje del lado del servidor, usar algún método para componer valores únicos.

Si queremos que el ancho de banda y la calidad de imagen tengan la misma importancia, debemos indicar valores numéricos para ambos parámetros.

```
cam.setQuality (20000,90);
```

El resto de las propiedades y los métodos, tanto para la webcam como para el micrófono, ya los hemos visto a lo largo de este capítulo.

Vincular webcam y audio a NetStream y enviarlo

Por último, lo que nos interesa saber es cómo vinculamos la cámara y el audio a un objeto **NetStream** a fin de transmitirlo al servidor:

```
ns = new NetStream(nc);
ns.attachCamera(cam);
ns.attachAudio(mic);
```

Como podemos ver, por medio del método **attachCamera()** incluimos la cámara y, a través del método **attachAudio()**, el micrófono. De esta manera, la instancia de la clase **NetStream** ya cuenta con la información que deseamos enviar al servidor. Lo único que nos queda por ver es de qué manera comenzar a enviar datos al servidor y cómo finalizar la conexión.

Publicar video

Por medio del método **publish()** enviamos la transmisión del video desde Flash a Flash Media Server 3. Los dos parámetros que debemos indicarle son:

- El **nombre del video**: no tenemos que indicarle la extensión, FMS3 generará automáticamente el video FLV.
- El **tipo de publicación**: utilizamos el tipo **"record"**, por medio del cual FMS3 publica y graba datos en vivo.

PUBLICAR ARCHIVOS MP4

Por medio del método **publish()** también podemos exportar videos en formato **MP4**. Para hacerlo, debemos anteponer la cadena **mp4:** al nombre del archivo y, a diferencia del formato **FLV** donde simplemente escribimos el nombre, también tenemos que indicarle la extensión del archivo. Por ejemplo, **ns.publish("mp4:nombre.mp4" , "record");**.

```
private function startRecording(event:MouseEvent):void{
     ns.publish("video" , "record");
}
```

Con estos dos parámetros que acabamos de definir en el código anterior, se grabará un video que se llamará **video.flv**.

TIPOS DE PUBLICACIÓN	DESCRIPCIÓN
"record"	Cuando utilizamos el tipo de publicación **"record"**, FMS3 va a publicar y grabar datos en vivo y almacenará el video bajo el nombre que hayamos definido en el parámetro **nombre**. Si ya existe un video con esa misma denominación, FMS3 lo reescribirá y reemplazará el archivo existente por el nuevo.
"append"	En caso de utilizar el modo de publicación **"append"**, FMS3 va a publicar y grabar datos en vivo y, a su vez, los guardará con el nombre que hayamos definido en el parámetro nombre.
"live"	Cuando utilizamos el modo de publicación **"live"**, FMS3 publica datos, pero no los graba.

***Tabla 1.** Tipos de publicación para el método **publish()**.*

Finalizar publicación

Si bien vimos que utilizamos el método **publish()** para comenzar a publicar un video, de algún modo debemos finalizar la grabación. Esto lo hacemos por medio del método **close()**, con el que terminamos la transferencia de un video.

```
private function stopRecording(event:MouseEvent):void{
     ns.close();
}
```

***Figura 22.** Los dos estados: durante la grabación y una vez que la detenemos. Mientras se graba un video, deshabilitamos el botón de grabar y quitamos su listener. Lo mismo hacemos con el botón de detener una vez que lo presionamos.*

Ubicación de los videos

Cuando publicamos contenidos por medio del método **publish()**, se generará un directorio en la carpeta que lleva por nombre el de nuestro proyecto, llamado **streams**. Dentro de **streams**, habrá otra carpeta denominada **_definst_** en la que se almacenarán los videos que se vayan publicando.

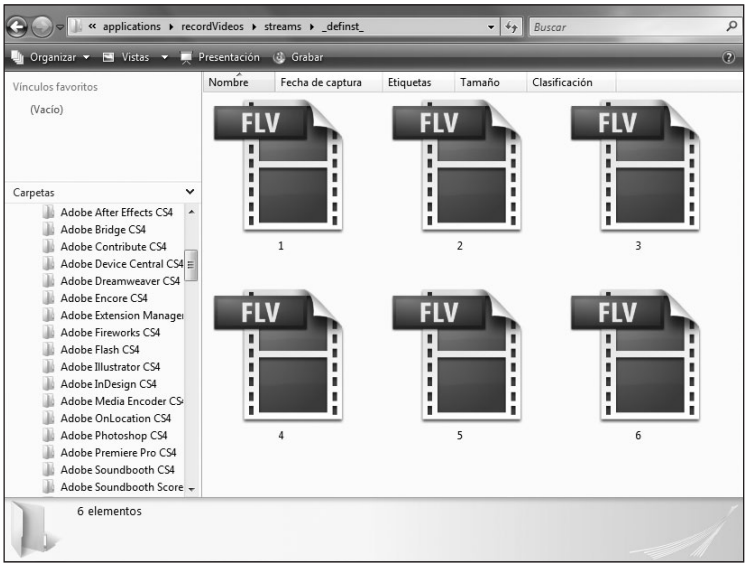


***Figura 23.** Los videos se almacenarán dentro de nombreDeAplicacion/streams/_definst_.*

Reproducir videos con streaming

Se tiende a confundir estos conceptos y considerar que son lo mismo, pero existe una serie de diferencias importantes entre la **carga progresiva** de contenidos y el **streaming**. En verdad, son dos conceptos totalmente distintos.

Cuando reproducimos un video del modo en el que lo hicimos en el capítulo anterior, estamos realizando una carga progresiva de los contenidos. Esto implica que llamamos al archivo del servidor desde el cliente y lo vamos descargando de forma

MÉTODO CLOSE() DE LAS CLASES NETSTREAM Y NETCONNECTION

Si bien las dos clases cuentan con el método **close()**, no debemos confundirlas. Al usar el método **close()** para un objeto **NetStream**, estamos dando por finalizada la transferencia de video entre cliente y servidor, pero no por esto se cierra la comunicación **RTMP** entre ambos. Para cerrar la comunicación entre **Flash** y **FMS3**, debemos utilizar el método **close()** de la clase **NetConnection**.

progresiva. El procedimiento es similar a la descarga de cualquier contenido vía HTTP, y a medida que vamos bajando el video, lo podemos reproducir.
El concepto de streaming es diferente y si bien en ambos casos los archivos son externos, en el streaming se establece una **conexión persistente** entre el cliente y el video. El **streaming**, a diferencia de la **carga progresiva**, nos permite acceder a información con la que podemos mejorar considerablemente la experiencia de navegación del usuario y optimizar la descarga de los contenidos.
Una de las principales ventajas de reproducir un video en **streaming** es que utilizamos menos memoria y espacio en el disco del cliente, debido a que no tiene que descargar el video (en la carga progresiva, sí se descarga el video). A su vez, se optimizan recursos, ya que solamente se descarga la parte del video que el cliente quiere ver y no la totalidad de los contenidos. También podemos acceder a información en tiempo real de la descarga y, en base a ella, tomar decisiones que repercutan en la experiencia del usuario. Por citar un ejemplo, si el usuario posee una baja tasa de transferencia, podemos modificar las propiedades del video en tiempo de ejecución a fin de optimizar la descarga. En definitiva, la carga a través de **streaming** tiene ciertas ventajas que es importante que conozcamos.
Ahora veremos un pequeño ejemplo para reproducir, en nuestra aplicación, el video que estamos grabando. No tiene sentido generar los controles que hacen al manejo de videos ya que dedicamos todo el capítulo anterior a ese tema. En esta aplicación, lo que hacemos es reproducir, en otra instancia de video, el video que vamos grabando desde nuestra webcam.

Figura 24. *Del lado izquierdo de la imagen, la entrada de video. Del lado derecho, la reproducción del video que estamos grabando.*

Cuando ejecutemos la película, veremos que la instancia de video que se encuentra a la izquierda posee mucha mejor calidad que la instancia de la derecha. Esto se debe, justamente, a que la primera responde a la captura de la webcam y la segunda es una

reproducción del video que estamos grabando. La instancia de la izquierda representa el **input** (la entrada) de video y la de la derecha, el **output** (la salida) de éste.
Reproducir un video en streaming es sencillo. En primer lugar, debemos crear la instancia del video que vamos a reproducir y utilizar el método **attachStream()** para asignarle el streaming a nuestra instancia:

```
nsIn=new NetStream(nc);
playVideo = new Video(cam.width, cam.height);
playVideo.x = 330;
playVideo.attachNetStream(nsIn);
addChild(playVideo);
```

Por último, para reproducir un video, tenemos que utilizar el método **play()**, al que le asignamos, como parámetro, el nombre del video que se está grabando.

```
nsIn.play(videoNumber.toString());
```

Si bien el streaming es superior a la carga progresiva de los contenidos y presenta una serie de ventajas sobre aquél, éste no es el único factor determinante a la hora de elegir una modalidad de descarga. Siempre debemos evaluar la magnitud del proyecto, su alcance, los recursos y los tiempos con los que contamos para llevar a cabo su desarrollo, y la cantidad de usuarios que utilizarán la aplicación, entre otras cosas. Al hacer este balance, veremos que, generalmente, la carga progresiva de los contenidos satisface nuestros requerimientos y evitamos la necesidad de instalar un servidor remoto, su configuración y la posterior adaptación de Flash. Nuestro objetivo es, como desarrolladores de Flash, conocer cómo manejar estas dos alternativas de descarga y saber cuándo optar por una y cuándo por otra.

... RESUMEN

Saber cómo acceder y manipular cámaras y micrófonos pone a nuestra disposición una enorme gama de posibilidades de desarrollo que actualmente están en auge y nos permiten un amplio espectro de alternativas, desde desarrollos experimentales a aplicaciones más complejas que requieran grabar audio y video a un servidor remoto. En el próximo capítulo veremos una nueva tecnología en relación con Flash, destinada a la creación de aplicaciones de escritorio: Adobe AIR.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1) ¿Para qué utilizamos el método estático **Microphone.getMicrophone()**?

2) ¿De qué manera podemos saber si el usuario cuenta con un micrófono conectado?

3) ¿Cómo detectamos si el usuario nos permitió acceder a su micrófono?

4) ¿Qué información nos proporciona la propiedad **activityLevel**?

5) ¿Qué método utilizamos para reproducir, por los parlantes, la entrada de sonido?

6) ¿Para qué se utilizan las propiedades **gain** y **rate**?

7) ¿Cómo accedemos a la cámara web de un sistema?

8) ¿Para qué se usa el método **setMode()**? ¿Qué parámetros le debemos indicar?

9) ¿Qué es Flash Media Server 3? ¿Para qué se utiliza?

10) ¿Con qué método grabamos un video al servidor y cuál utilizamos para reproducir un video en streaming?

EJERCICIOS PRÁCTICOS

1) Genere una aplicación que utilice la cámara web en la que se evalúe si el usuario cuenta con una cámara y si permite el acceso a ella.

2) Haga lo mismo para el micrófono.

3) Cree una aplicación experimental con la información que le brindan la cámara web y el micrófono.

4) Tome imágenes de la cámara empleando la clase **BitmapData**.

5) Genere una aplicación que grabe videos utilizando Flash Media Server 3, RED5 o cualquier servidor que le permita la comunicación por medio del protocolo RTMP entre cliente y servidor

Aplicaciones de escritorio

Colaborador:
Fabricio Mouzo

Integrar Flash con Adobe AIR nos permite crear aplicaciones de escritorio usando la sintaxis que ya hemos aprendido. Para familiarizarnos con este entorno de desarrollo vamos a generar un navegador web y modificaremos el reproductor de MP3 que hicimos anteriormente para convertirlo en una aplicación de escritorio que reproduzca archivos almacenados en nuestro equipo.

SERVICIO DE ATENCIÓN AL LECTOR: usershop@redusers.com

FLASH Y ADOBE AIR

Antes de adentrarnos en un desarrollo que integra Flash con **Adobe AIR** para generar aplicaciones de escritorio, debemos hacer un repaso por las cuestiones de mayor importancia de esta nueva plataforma, que nos brinda una enorme gama de posibilidades, muchas de las cuales las iremos viendo a lo largo de este capítulo.

¿Qué es Adobe AIR?

Adobe AIR (o **Adobe Integrated Runtime**) es un entorno de ejecución o **runtime** (**tiempo de ejecución versátil**) que nos permite salir de las limitaciones que poseen las aplicaciones web para crear **RIAs** que funcionen como un programa tradicional de escritorio, interactuando con sus contenidos.
Las aplicaciones desarrolladas con AIR requieren ser **empaquetadas**, **firmadas digitalmente** y, por último, **instaladas** dentro del sistema de archivos local del usuario. Una vez instalada la aplicación, podemos tener acceso al sistema de archivos brindando una mayor flexibilidad para la funcionalidad de nuestras aplicaciones.

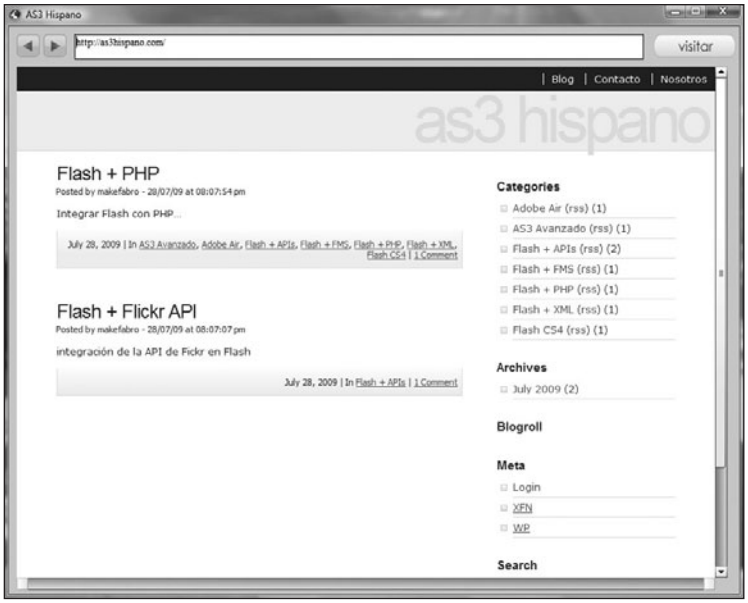


Figura 1. *Con Adobe AIR podemos desarrollar un navegador web, como veremos en este capítulo.*

Lo más interesante de AIR es que nos permite implementar todo lo que hemos aprendido hasta ahora de ActionScript 3.0 para crear aplicaciones sofisticadas que interactúen con el sistema operativo e Internet sin necesidad de conocer más que la utilización de algunas clases que nos provee la API de AIR.

Cuando generamos una aplicación AIR desde Flash, utilizamos un método distinto de publicación al que estamos acostumbrados. En lugar de crear un archivo con extensión SWF, generaremos un archivo **.AIR**, que le permitirá al usuario realizar la instalación de la aplicación que hayamos desarrollado en su sistema local.

Instalación de AIR

Para crear nuestras aplicaciones de escritorio, no es necesario realizar ninguna instalación, ya que el entorno de desarrollo de AIR está incorporado a Flash CS4. Sin embargo, así como para poder visualizar los archivos SWF en un navegador web debemos tener instalado el plugin de Flash Player, para instalar aplicaciones AIR en una computadora debemos poseer el runtime de AIR.

Para obtener este runtime, debemos ir al sitio **http://get.adobe.com/es/air/** y presionar el botón **Descargar ahora**. Luego, una vez realizada la descarga del archivo, simplemente lo ejecutamos y seguimos las instrucciones del instalador, que nos va guiando a lo largo del sencillo proceso de instalación.

Figura 2. *Al descargar Adobe AIR, debemos elegir el sistema operativo y el idioma que necesitamos.*

COMPATIBILIDAD CON SISTEMAS OPERATIVOS

Una de las grandes ventajas de **Adobe AIR** es que las aplicaciones que desarrollamos no necesitan adaptación alguna para su implementación en los diferentes sistemas operativos. Funcionan perfectamente en **Windows 2000**, **Windows XP**, **Windows Vista**, **Mac OSX** y **Linux**.

API de AIR

Como hemos visto en anteriores capítulos, las **APIs** son un conjunto de funciones de las que hacemos uso sin necesidad de programar sus contenidos. Adobe AIR nos brinda su propia **API** para poder interactuar con el sistema operativo y desarrollar aplicaciones que **creen**, **modifiquen**, **copien** o **eliminen** archivos del sistema local. También podremos incorporar, en nuestras aplicaciones, todas las funcionalidades básicas que posee un programa de escritorio tradicional, como la posibilidad de **minimizar**, **maximizar**, **cerrar** o **arrastrar** archivos (**draguear**) desde una carpeta para que nuestras aplicaciones las utilicen; o crear **bases de datos** locales con **SQL Lite** para el almacenamiento de información a nivel local. Las posibilidades de Adobe AIR son muchísimas y algunas las iremos conociendo a lo largo de este capítulo.

CREAR UN NAVEGADOR WEB CON FLASH Y AIR

Para poner en práctica la integración de Flash y AIR, comenzaremos llevando a cabo el desarrollo de un sencillo **navegador**, que contará con las características típicas de cualquier aplicación de escritorio y nos permitirá acceder y ver cualquier sitio web. Este ejemplo nos dará la posibilidad de apreciar el potencial de la clase **`HTMLLoader`** al utilizarla desde una aplicación desarrollada en Adobe AIR.

***Figura 3.** Desde **http://kidrocket.org/** podemos descargar un excelente ejemplo de un navegador pensado para niños, desarrollado íntegramente con Adobe AIR.*

Iniciar un nuevo proyecto

Lo primero que debemos hacer para comenzar a desarrollar cualquier aplicación de escritorio es crear un archivo .FLA, que nos permita exportar mediante el entorno de AIR. Al abrir el programa, podemos seleccionar la opción **Archivo de Flash (Adobe AIR)** o dirigirnos a **Archivo/Nuevo** y allí seleccionar **Archivo de Flash (Adobe AIR)**.

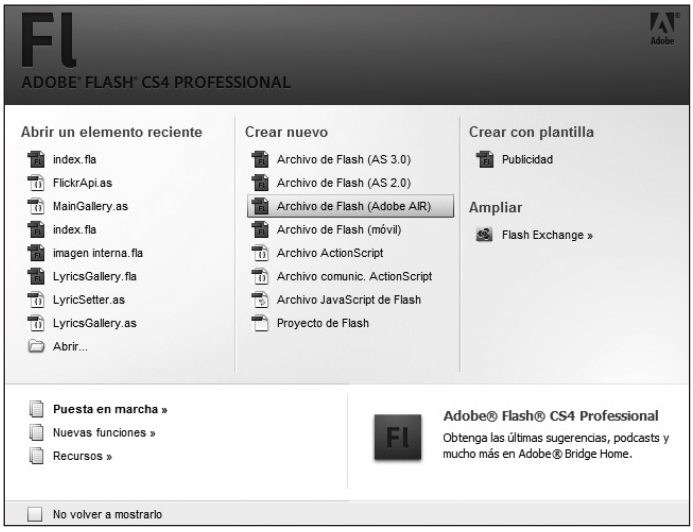


***Figura 4.** Podemos seleccionar desde **Crear nuevo** la opción **Archivo de Flash (Adobe AIR)**. Ésta está disponible solamente en **Flash CS4**.*

Comenzar nuestro desarrollo

Una vez creado el archivo, estamos listos para empezar a trabajar en nuestro navegador. Lo primero que debemos hacer es generar los **botones** de nuestra interfaz gráfica y exportarlos con sus respectivas instancias para poder usarlos desde nuestro código. No nos detendremos en esto ya que, siendo el último capítulo, no deberíamos tener problemas para llevar a cabo este proceso, pero, si hay una cuestión que nos interesa, en el **capítulo 7** tuvimos nuestro primer acercamiento a la utilización de los **Componentes de Flash** implementando el reproductor de videos **FLVPlayback**. En

CENTRO DE AYUDA DE LA API DE AIR

En **http://help.adobe.com/es_ES/AIR/1.1/devappsflash/** encontramos el centro de ayuda de AIR en español. Éste cuenta con información detallada sobre cada una de las clases de la **API** de **AIR**, respecto a su instalación, configuración, recursos y mucho más. Por eso es importante tener este sitio a mano al llevar a cabo cualquier desarrollo que implique el uso de Adobe AIR.

esta sección, haremos uso del componente **UIScrollBar**, por medio del cual podemos generar un **scroll** en nuestra aplicación. Si bien en general creamos nuestras propias clases o utilizamos desarrollos de terceros en lugar de componentes, no es éste un tema central que haga a nuestro desarrollo, por lo que en esta ocasión aplicaremos un componente de Flash para darle vida al scroll de nuestro navegador.
Para poder usar este componente tenemos que guardar una instancia de él dentro de nuestra biblioteca. Para ello, debemos tener el panel **COMPONENTES** abierto (en caso de no contar con este panel, podemos abrirlo presionando **CTRL+F7** o desde el menú **Ventana/Componentes**). Desde allí, arrastramos el componente que vamos a usar hasta el panel **BIBLIOTECA**; en nuestro caso: **UIScrollBar**.

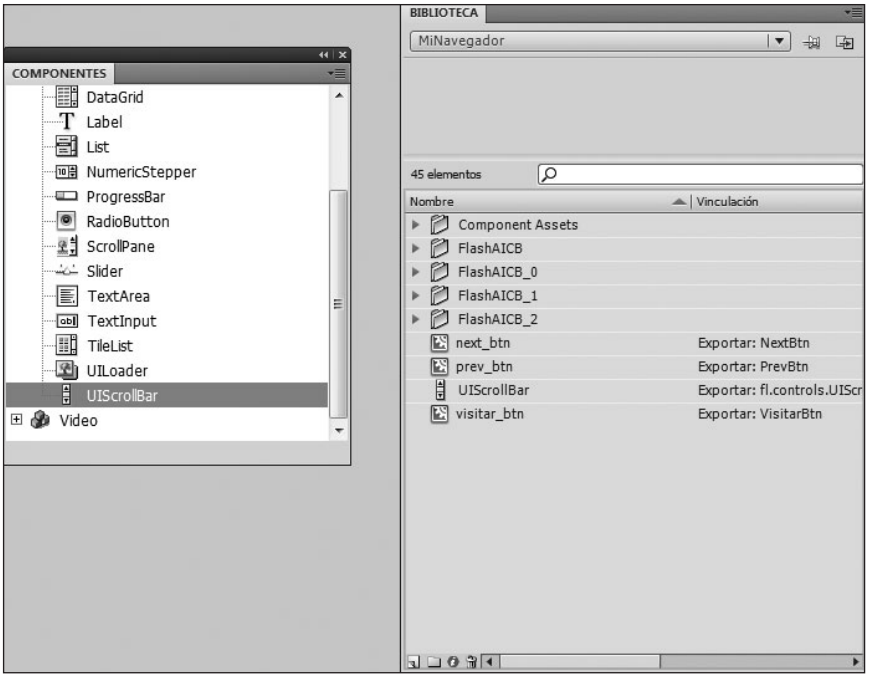


***Figura 5.** Al guardar el componente UIScrollBar dentro de la biblioteca, podremos acceder a él desde el código que veremos en las próximas páginas.*

III TESTEAR APLICACIÓN EN AIR MIENTRAS LA DESARROLLAMOS

A medida que desarrollamos nuestras aplicaciones, podremos visualizarlas desde Flash del mismo modo que lo hacemos con nuestros archivos SWF: desde el menú **Control/Probar película** o con el atajo **CTRL+ENTER**. Esto es verdaderamente útil ya que no tenemos que publicar el archivo cada vez que queramos probar alguna funcionalidad de nuestra aplicación de escritorio.

Próximamente veremos cómo hacer uso del componente **UIScrollBar**; por el momento, lo que nos interesa es que éste se encuentre en nuestra biblioteca junto al resto de los elementos que conforman la interfaz de nuestro navegador. Una vez que contamos con eso preparado, podemos comenzar a pensar en el desarrollo de nuestra aplicación.
Como ya hemos visto en los capítulos anteriores, es posible programar desde la IDE de Flash o generar una clase para vincular al SWF. Si bien en este ejemplo trabajaremos con una clase, en caso de querer adaptarlo a la IDE, simplemente debemos quitar la sintaxis que hace al funcionamiento de las clases y disponer el componente en el primer fotograma de nuestro archivo .FLA.

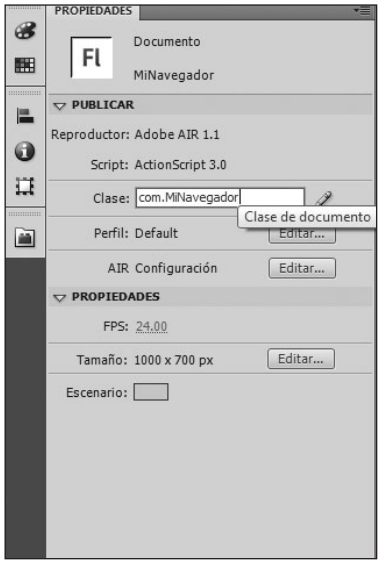


***Figura 6.** El modo en el que se vincula una clase a un archivo **FLA** es exactamente el mismo, independientemente de que se trate de una aplicación de escritorio (**AIR**) o de una película **SWF** convencional.*

Comencemos con el código. Para nuestro navegador haremos uso de algunas clases nuevas que no hemos visto hasta aquí y varias de ellas son de uso exclusivo

MDM ZINC 3.0

Adobe AIR no fue el primer entorno de desarrollo creado para generar aplicaciones de escritorio en Flash. En el año 2002, la empresa **Multimedia Limited** creó **Zinc**, un entorno de desarrollo similar a **AIR** que con el tiempo maduró notablemente y en la actualidad brinda una enorme cantidad de posibilidades. Su desventaja es que el servicio no es gratuito.

del entorno de desarrollo que propone Adobe AIR, como iremos viendo a lo largo de este capítulo. Veamos el código a continuación:

```
package com{
     import flash.display.Sprite;
     import flash.html.HTMLLoader;
     import flash.net.URLRequest;
     import flash.display.NativeWindow
     import flash.events.*;
     import flash.text.TextField;
     import flash.text.TextFieldType;
     import flash.text.TextFormat;
     import fl.controls.ScrollBar;
     import fl.events.ScrollEvent;
```

Una vez importadas las clases que necesitaremos, realizamos la declaración de las variables que utilizaremos para generar la **GUI** (**interfaz gráfica del usuario**):

```
public class MiNavegador extends Sprite{
     private var urlReq:URLRequest;
     private var miHtml:HTMLLoader;
     private var direccion_txt:TextField;
     private var visitar_btn:VisitarBtn;
     private var next_btn:NextBtn;
     private var prev_btn:PrevBtn;
     private var scrollVertical:ScrollBar;
     private var scrollHorizontal:ScrollBar;
```

Declaradas las variables que usaremos en nuestro código, dentro del constructor de nuestra clase llamaremos a la función **crearGUI()**, donde asignaremos todos los elementos de la interfaz gráfica, como vemos en las líneas siguientes:

```
public function MiNavegador():void{
     crearGUI();
}
```

No analizaremos la sintaxis básica de la función **crearGUI()** que hace al armado de la interfaz; simplemente nos abocaremos a los nuevos conceptos que abarca este capítulo.

Clase HTMLLoader

Utilizaremos la clase **HTMLLoader** cada vez que necesitemos incluir **contenido HTML** dentro de nuestras aplicaciones. Ya conocemos a la perfección la manera en la que generamos instancias para nuestras clases. No debería haber nada que no conozcamos del siguiente código:

```
miHtml = new HTMLLoader();
miHtml.x=10;
miHtml.y= 50;
miHtml.width = stage.stageWidth - 40;
miHtml.height = stage.stageHeight - 70;
addChild(miHtml);
miHtml.addEventListener(Event.COMPLETE, cargaCompleta, false, 0, true);
miHtml.load(new URLRequest("http://as3hispano.com/"));
```

Una vez que añadimos nuestra instancia de la clase **HTMLLoader** (**miHtml**) al escenario, tenemos todo lo necesario para realizar la descarga de contenido HTML dentro de él. Un buen modo de pensar la clase **HTMLLoader**, es en similitud a la clase **Video** que hemos visto en el **capítulo 7**. Lo que hacemos por medio de ella es generar el **espacio** dentro del cual se cargarán los contenidos. Ahora bien, veamos de qué manera cargarlos. Generaremos dos funciones que se ocuparán de cargar los contenidos dentro de **HTMLLoader**. A estas funciones las llamaremos **visitarHtml()** y **visitarHtmlConEnter()**. La primera de ellas será en respuesta a presionar el botón **visitar** de la interfaz de nuestro navegador:

```
visitar_btn = new VisitarBtn();
visitar_btn.x = stage.stageWidth - visitar_btn.width - 5;
visitar_btn.y = 10;
visitar_btn.buttonMode = true;
addChild(visitar_btn);
visitar_btn.addEventListener(MouseEvent.MOUSE_DOWN, visitarHtml);
```

La segunda responde al **ENTER** de nuestro teclado, ya que lo común es presionar esta tecla una vez que se termina de escribir la dirección; los navegadores web nos han acostumbrado a esto. Veamos cómo lograrlo:

```
stage.addEventListener(KeyboardEvent.KEY_DOWN,visitarHtmlconEnter, false, 0,
true);
```

Lo importante de estas dos funciones es que notemos el uso del método **load()** de la clase **HTMLLoader**, como vemos en el siguiente código:

```
private function visitarHtml(e:MouseEvent):void{
     urlReq = new URLRequest(direccion_txt.text);
     miHtml.load(urlReq);
}

private function visitarHtmlconEnter(e:KeyboardEvent):void{
     if(e.keyCode == 13){
     urlReq = new URLRequest(direccion_txt.text);
     miHtml.load(urlReq);
}
```

Sea con un **ENTER** o con un clic sobre un **MovieClip**, con estas funciones estamos tomando el valor de la cadena de texto que se haya ingresado en el campo de texto **direccion_txt** y se la asignamos como parámetro a una instancia de la clase **URLRequest**, que ya hemos visto a lo largo de este libro. Luego, mediante el método **load()** de la clase **HTMLLoader**, realizamos la descarga de dicha URL.

***Figura 7.** Al presionar el botón visitar o la tecla ENTER, cargamos la URL que se encuentra en el campo de texto de nuestro navegador. La carga se hace por medio del método load() de la clase URLLoader.*

CLASE HTMLLOADER: DIFERENCIAS ENTRE SWF Y AIR

Si bien la clase **HTMLLoader** nos permite descargar contenidos **HTML** en archivos **SWF** y **AIR**, existen importantes diferencias: mientras en un **SWF** es imposible renderizar todos los tags HTML, no hay soporte para **CSS**, ni se puede comunicar con **JavaScript**; **AIR** realiza todas estas tareas sin limitación alguna, lo que nos permite llevar a cabo el desarrollo de nuestro navegador en **AIR**.

Mejorar la experiencia de navegación

Para poder brindar un mayor nivel de experiencia al usuario, le asignaremos a nuestro navegador web la posibilidad de recorrer el **historial** de páginas visitadas. Para ello, utilizaremos los métodos **historyBack()** y **historyForward()**, propios de la clase **HTMLLoader**. En primer lugar, debemos asignarles los correspondientes eventos a los botones de nuestra interfaz para navegar en el historial:

```
prev_btn = new PrevBtn();
prev_btn.x = 10;
prev_btn.y = 10;
prev_btn.buttonMode = true;

addChild(prev_btn);
prev_btn.addEventListener(MouseEvent.MOUSE_DOWN, irPrev);

next_btn = new NextBtn();
next_btn.y = 10;
next_btn.buttonMode = true;
next_btn.x = prev_btn.x + prev_btn.width + 5;

addChild(next_btn);
next_btn.addEventListener(MouseEvent.MOUSE_DOWN, irNext);
```

Al presionar **prev_btn**, llamaremos a la función **irPrev()**. Dentro de ella, por medio del método **historyBack()**, podemos volver a la página o a las páginas que visitamos anteriormente. Lo hacemos de la siguiente manera:

```
private function irPrev(e:MouseEvent):void{
     miHtml.historyBack();
}
```

III OTRAS OPCIONES PARA DESARROLLO EN AIR

No sólo podemos crear aplicaciones AIR con **Flash CS3** y **CS4**. Adobe también permite el desarrollo de aplicaciones de escritorio con **Flex 3**, **HTML**, **AJAX** o **JavaScript**. Si deseamos utilizar alguna de éstas tecnologías, deberemos descargar un **kit de desarrollo** que brinda Adobe gratuitamente en su página web de descargas.

Y al presionar **next_btn**, llamaremos a la función **irNext()**, dentro de la cual hacemos uso del método **historyForward()**. Éste nos permite volver a un nivel superior luego de haber hecho uso del método **historyBack()**:

```
private function irNext(e:MouseEvent):void{
     miHtml.historyForward();
}
```

Así, a través de los métodos **historyBack()** e **historyForward()**, podemos recorrer el historial de sitios visitados. Ejecutamos estos métodos presionando las flechas **atrás** y **adelante** de nuestro navegador.
Si bien es verdaderamente simple hacer uso de estos métodos, es importante notar que necesitamos actualizar el historial cada vez que realicemos una descarga de una nueva URL. Al hacerlo, recordemos que al generar la instancia de la clase **HTMLLoader**, le asignamos un listener a fin de detectar cuándo se ha completado la carga. Entonces, ingresamos lo siguiente:

```
miHtml.addEventListener(Event.COMPLETE, cargaCompleta, false, 0, true);
```

Como vemos, la función **cargaCompleta()** realiza varias acciones, pero la que nos interesa es el llamado a la función **actualizarHistorial()**. El resto de los contenidos de esta función los conoceremos próximamente.

```
private function cargaCompleta(e:Event):void{
     actualizarScrolls();
     actualizarHistorial();
     direccion_txt.text = miHtml.window.location;
     stage.nativeWindow.title = miHtml.window.document.title;
}
```

III ¿QUIÉNES USAN ADOBE AIR?

Muchas de las empresas más grandes del mundo que desarrollan actividades comerciales a través de sus sitios web elijen **Adobe AIR** para la creación de aplicaciones de escritorio. Por citar tan sólo dos ejemplos, **AOL** utilizó Adobe AIR para generar una aplicación de videos y **eBay** generó una aplicación para realizar todas las operaciones de su sitio web desde el escritorio.

La función **actualizarHistorial()** es la que se ocupa de actualizar el historial de nuestra navegación cada vez que se produce una nueva carga de un sitio web. Conociendo el valor actual de la propiedad **historyPosition** de la clase **HTMLLoader()**, podemos saber cuántos sitios se han visitado y asignarles comportamientos a los botones de navegación en base a esta información:

```
private function actualizarHistorial():void{
     if(miHtml.historyPosition == 0){
            prev_btn.enabled = false;
            prev_btn.alpha = .5;
     }else{
            prev_btn.enabled = true;
            prev_btn.alpha = 1;
     }
     if(miHtml.historyPosition == (miHtml.historyLength -1)){
            next_btn.enabled = false;
            next_btn.alpha = .5;
     }else{
            next_btn.enabled = true;
            next_btn.alpha = 1;
     }
}
```

En primer lugar, si la posición actual es igual a **0**, sabemos que no hay un nivel inferior, por lo que deshabilitamos el botón para ir al sitio previo (**prev_btn**):

```
if(miHtml.historyPosition == 0){
     prev_btn.enabled = false;
     prev_btn.alpha = .5;
}
```

En caso contrario, de no ser **0** la posición actual, sabemos que hay al menos un nivel inferior; y en este caso, habilitamos el botón **prev_btn**:

```
else{
     prev_btn.enabled = true;
     prev_btn.alpha = 1;
}
```

La lógica es la misma para el botón **next_btn** pero, en este caso, lo que debemos averiguar es si la posición actual es la **última** dentro del historial. Esto lo podemos conocer controlando, por medio de un **condicional**, los valores de la propiedad **historyPosition** y de una propiedad que no hemos visto hasta aquí: **historyLength**. Por medio de la propiedad **historyLenght**, podemos saber la cantidad total de sitios que se han almacenado en el historial. En base al siguiente condicional, es posible determinar si nos encontramos en la última posición. De ser así, nuestro botón para pasar al sitio siguiente no cumple ninguna función, entonces lo deshabilitamos:

```
if(miHtml.historyPosition == (miHtml.historyLength -1)){
     next_btn.enabled = false;
     next_btn.alpha = .5;
}
```

Si, en cambio, no nos hallamos en la última posición del historial, eso implica que hay al menos un nivel superior. En este caso, habilitamos el botón:

```
else{
     next_btn.enabled = true;
     next_btn.alpha = 1;
}
```

***Figura 8.** Si conocemos el **historial de navegación**, podemos mejorar el diseño y la funcionalidad de nuestro navegador, habilitando o deshabilitando los botones para ir hacia atrás o hacia adelante.*

Utilización del componente ScrollBar

Anteriormente, hicimos hincapié en la necesidad de incluir el componente **UIScrollBar** dentro de nuestra biblioteca. Vamos a tener que hacer uso de él debido a la diferencia en las dimensiones de los sitios web. Al diseñar un sitio, son los desarrolladores quienes definen las dimensiones en alto y ancho de los contenidos. Esto hace que existan sitios en las más diversas **resoluciones**, por lo que vamos a tener que asignar barras para **hacer scroll** sobre los contenidos que cargue nuestro navegador. Esto sucederá en aquellos casos en los que el tamaño del contenido que se descargue dentro de nuestro **HTMLLoader** sea mayor al de las dimensiones de la propia instan-

cia de la clase **HTMLLoader** que hemos creado. A continuación, veremos la sintaxis necesaria para implementar el componente **UIScrollBar** en nuestra aplicación:

```
scrollVertical = new ScrollBar();
scrollVertical.direction = “vertical”;
scrollVertical.x = miHtml.x + miHtml.width;
scrollVertical.y = miHtml.y;
scrollVertical.height = miHtml.height;
scrollVertical.lineScrollSize = 20;
scrollVertical.pageSize = miHtml.height;
addChild(scrollVertical);
scrollVertical.addEventListener(ScrollEvent.SCROLL, vScroll, false, 0,
true);
```

Conocemos varias de las propiedades que estamos empleando en este código (**x**, **y**, **height**), pero veremos que hay otras que son propias del componente **UIScrollBar**:

- Propiedad **direction**: esta propiedad nos da la posibilidad de asignar una dirección por medio de una cadena de texto; **“horizontal”** o **“vertical”** para el scroll en cada uno de los sentidos. En nuestro caso, empleamos un scroll en sentido vertical, y para eso usamos: **scrollVertical.direction = “vertical”;**.
- Propiedad **lineScrollSize**: nos permite asignar la cantidad de líneas en las que se desplazará nuestro scroll hacia arriba o hacia abajo en caso de ser un scroll vertical y hacia la derecha o la izquierda en caso de ser un scroll horizontal. Definimos su valor en **20**, un número razonable para este tipo de scroll ya que nos posibilita una visualización fluida de los contenidos: **scrollVertical.lineScrollSize = 20;**.
- Propiedad **pageSize**: esta propiedad obtiene o define el número de líneas que contiene una página. Para definir este valor acorde al contenido que se cargará, debemos igualarlo con el alto de nuestra instancia de la clase **HTMLLoader** de la siguiente manera: **scrollVertical.pageSize = miHtml.height;**.

Utilizando el listener **ScrollEvents**, podremos detectar cuándo se está realizando scroll sobre los contenidos, como vemos a continuación:

```
scrollVertical.addEventListener(ScrollEvent.SCROLL, vScroll);
```

Accediendo a la propiedad **position** de nuestro evento, sabremos la posición a la cual debemos hacer scroll en los contenidos. Este valor se lo asignamos a la propiedad **scrollV** de nuestra instancia (**miHtml**) de la clase **HTMLLoader**. Como **scrollV**

es una propiedad de lectura y escritura, podemos aplicarle una posición del desplazamiento vertical de un contenido HTML:

```
private function vScroll(e:ScrollEvent):void{
     miHtml.scrollV = e.position;
}
```

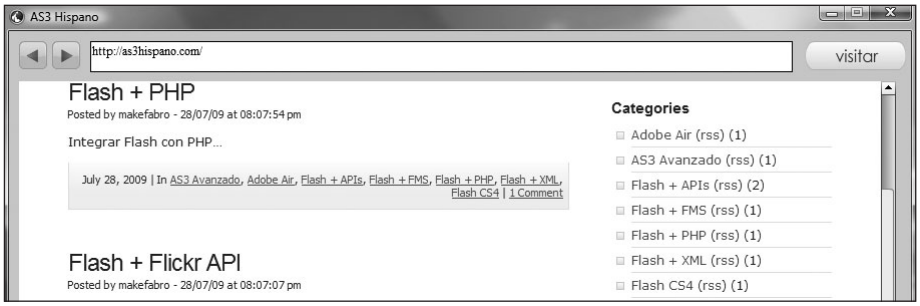


Figura 9. *Cuando el contenido de un sitio supera las dimensiones de nuestro navegador, hacemos uso del componente* ***UIScrollBar*** *para poder hacer scroll y ver la totalidad de los contenidos.*

Clase NativeWindow

Para finalizar con nuestro navegador, haremos uso de la clase **NativeWindow** de Adobe AIR. Por medio de esta clase, podemos cambiar el **título de la ventana** nativa de nuestra aplicación AIR. En nuestro ejemplo, reemplazaremos el nombre de la ventana por el título del sitio web que estamos visitando. Para hacerlo, debemos utilizar la propiedad **title** que nos proporciona la clase:

```
stage.nativeWindow.title = miHtml.window.document.title;
```

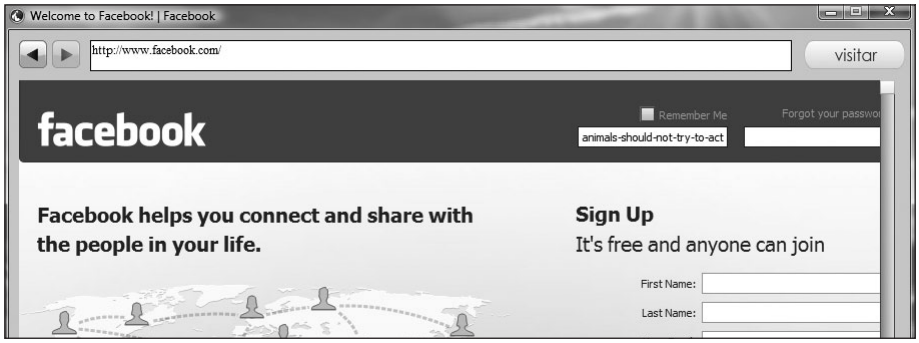


Figura 10. *El título de nuestra aplicación cambiará por el título del sitio que estemos visitando. Para esto, utilizamos la propiedad* ***title*** *de la clase* ***NativeWindow.***

Publicar la aplicación

Una vez concluido el desarrollo, debemos publicar la aplicación, es decir, **generar el instalador**. Al publicar un archivo FLASH (AIR), crearemos un archivo **.AIR**, que será el instalador de nuestra aplicación. Es importante leer con atención las siguientes páginas ya que veremos que hay una considerable cantidad de **opciones** que son realmente útiles y que vale la pena que conozcamos a fin de mejorar la apariencia y el funcionamiento de nuestra aplicación. Para publicar nuestro desarrollo, dentro de Flash debemos ir a **Archivo/Configuraciones de AIR...**. También podemos acceder a la ventana de configuración desde la barra **PROPIEDADES**, presionando el botón **Editar...**, donde dice **AIR Configuración...**.

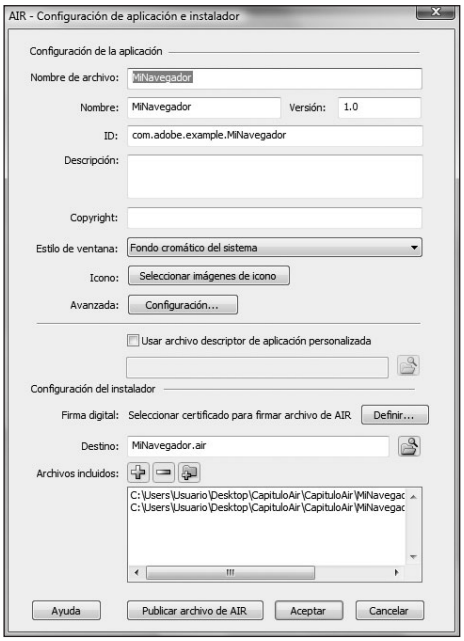


Figura 11. *Ventana AIR - Configuración de aplicación e instalador.*

Veamos cada una de las opciones:

- **Nombre de archivo:** debemos indicar el nombre del archivo de nuestra aplicación. Por defecto, éste será la denominación que lleva nuestro archivo **SWF**.
- **Nombre:** éste será el que utilizará el instalador para generar un archivo de aplicación y su respectiva carpeta durante el proceso de instalación en el sistema operativo del usuario. En caso de no definirlo, llevará el nombre del archivo **SWF**.
- **Versión:** este campo es opcional; se puede utilizar para especificar el número de la versión de Adobe AIR que se ha utilizado para desarrollar la aplicación.
- **ID:** sirve para identificar nuestra aplicación con un ID único. Si deseamos, podremos cambiar el ID que viene por defecto, aunque no es necesario hacerlo.

- **Descripción:** este campo es opcional y nos permite introducir una breve explicación de nuestra aplicación, que se mostrará durante la instalación.
- **Copyright:** este campo nos da la posibilidad de ingresar un aviso de derechos reservados que se mostrará durante la instalación. Es opcional y por defecto estará vacío.
- **Estilo de ventana:** con esta opción podremos seleccionar entre tres tipos de estilos de ventana para nuestra aplicación:

  - **Fondo cromático del sistema:** es el valor por defecto a la hora de exportar una aplicación de Adobe AIR. Dejando esta opción seleccionada, se utiliza el estilo de las ventanas estándar del sistema operativo. Este fondo es el que hemos usado a lo largo del desarrollo de nuestro navegador web en AIR.

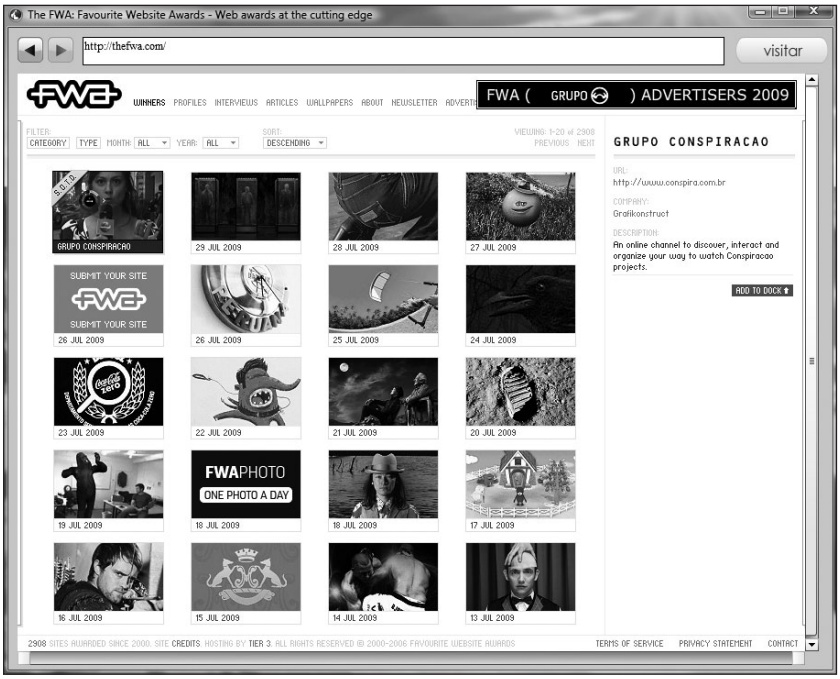


***Figura 12.** Ejemplo de ventana estándar del sistema operativo, con sus respectivas funciones para **minimizar**, **maximizar** y **cerrar**.*

NOMBRE E ID

El nombre que le demos a nuestra aplicación deberá contener solamente aquellos caracteres permitidos por el sistema operativo para nombres de archivos o carpetas. En cuanto al identificador (**ID**), solamente es posible utilizar los caracteres **0-9**, **a-z**, **A-Z**, **.** (**punto**) y **–** (**barra media**), y su longitud debe ser entre 1 a 212 caracteres.

- **Fondo cromático personalizado (opaco)**: seleccionando esta opción, eliminamos la ventana estándar del sistema operativo para nuestra aplicación. En caso de querer hacer uso de los botones para minimizar, maximizar y cerrar en nuestra aplicación, tendremos que programar sus comportamientos. Su principal ventaja es que nos permite lograr una mayor personalización y unidad de diseño en nuestras aplicaciones.

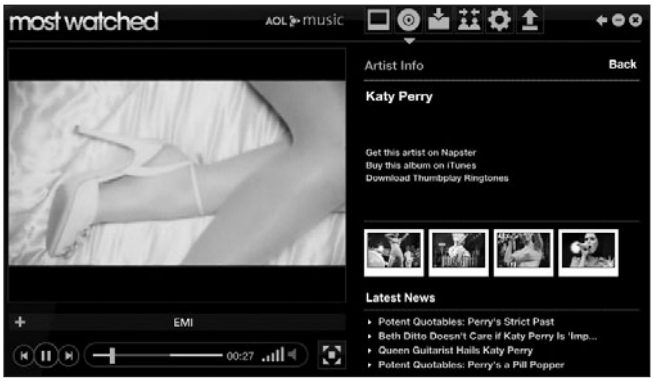


Figura 13. *La aplicación de los videos de* ***AOL*** *es un claro ejemplo de este tipo de ventana y evidencia un mejor acabado y diseño.*

- **Fondo cromático personalizado (transparente)**: esta opción nos brinda la misma funcionalidad que un fondo cromático personalizado opaco, pero también nos permite hacer uso de ventanas transparentes; en nuestro próximo ejemplo, veremos de qué manera utilizar estas ventanas. Al igual que en el tipo de ventana que vimos anteriormente, nos da mayores posibilidades en lo que a diseño de interfaz se refiere, logrando una mayor personalización.

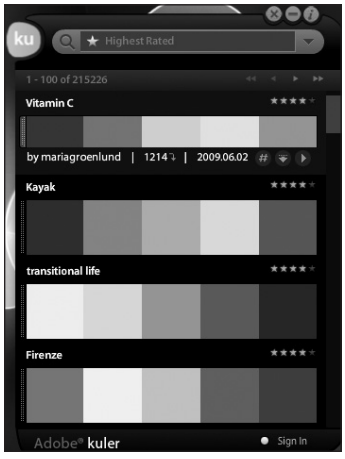


Figura 14. Adobe Kuler *utiliza una interfaz con ventana transparente; lo que permite un acabado completamente distinto al que brindan las aplicaciones tradicionales.*

- **Icono:** al hacer clic en el botón **Seleccionar imágenes de icono** se nos abrirá una nueva ventana donde podremos seleccionar un icono personalizado para nuestra aplicación. En este caso, podemos seleccionar cuatro versiones en formato **PNG** en las siguientes dimensiones: 128 x 128, 48 x 48, 32 x 32 y 16 x 16.

Figura 15. *Desde la ventana AIR - Imágenes de icono, debemos seleccionar las rutas correspondientes a cada uno de los iconos en sus cuatro formatos.*

- **Avanzada:** dentro de este panel encontraremos dos secciones importantes. **Tipos de archivos asociados** nos permite especificar los tipos de archivo que manejará nuestra aplicación, es decir, definimos qué tipos de archivo se abrirán con nuestro programa. Por ejemplo, si queremos que las imágenes JPG, PNG y GIF las abra nuestro programa, debemos indicarlo aquí. En **Configuración de la ventana inicial** podremos definir un tamaño y una posición inicial para nuestra pantalla e información importante en lo que hace al uso de aplicaciones de escritorio: si la aplicación se puede maximizar, minimizar, redimensionar, etcétera. La siguiente lista presenta todas las opciones disponibles:

ARCHIVOS ASOCIADOS

Debemos seleccionar cuidadosamente los tipos de archivo que deseemos asociar, ya que, al realizar este procedimiento, todos los archivos del sistema se abrirán con nuestra aplicación. Si bien esto es una gran ventaja que nos brinda Adobe AIR, también es una gran responsabilidad ya que en el proceso podemos dañar archivos e incluso generar problemas en el sistema operativo del usuario.

PROPIEDAD	DESCRIPCIÓN
Anchura	Especifica la anchura inicial de la ventana. Por defecto, este campo se encuentra vacío.
Altura	Determina la altura inicial de la ventana. Por defecto, este campo se encuentra vacío.
X	Establece la posición inicial de la ventana en sentido horizontal. Por defecto, este campo se encuentra vacío.
Y	Asigna la posición inicial de la ventana en sentido vertical. Por defecto, este campo se encuentra vacío.
Anchura máxima y Altura máxima	Definen el tamaño máximo de la ventana. Campos vacíos por defecto.
Anchura mínima y Altura mínima	Definen el tamaño mínimo de la ventana. Campos vacíos por defecto.
Se puede maximizar	Podemos definir si el usuario podrá maximizar la ventana. Por defecto, esta opción está permitida.
Se puede minimizar	Podemos establecer si el usuario podrá minimizar la ventana. Por defecto, esta opción está disponible.
Redimensionable	Podemos definir si el usuario podrá cambiar el tamaño de la ventana. Por defecto, esta opción está permitida.
Visible	Por medio de este parámetro establecemos si la aplicación será visible apenas se inicie. Su valor predeterminado es **true**.

Tabla 1. *Propiedades disponibles para la configuración de la ventana inicial de una aplicación AIR.*

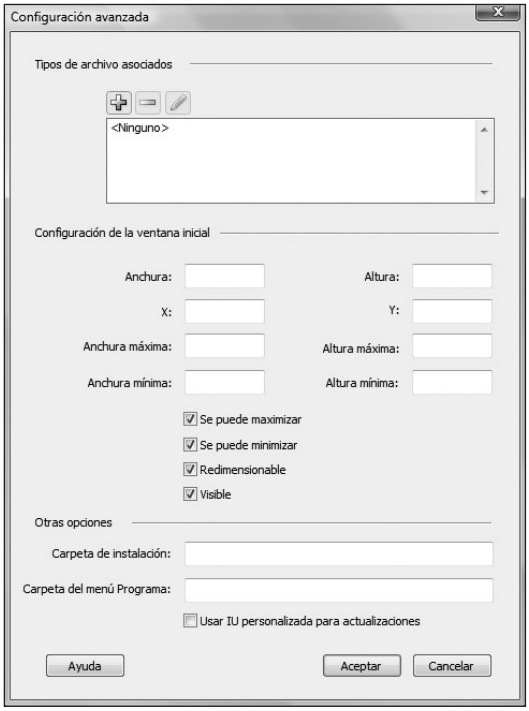


Figura 16. *Ventana* ***Configuración avanzada****. Desde aquí podemos definir comportamientos que son importantes para una aplicación de escritorio.*

- **Usar archivo descriptor de aplicación personalizada**: por medio de esta opción podemos utilizar un **archivo XML** que describa nuestra aplicación. En caso de no usarla, se empleará por defecto la descripción que utiliza Adobe Air, que se guarda en el archivo **nombre_aplicación-app.xml**.
- **Firma digital:** todas las aplicaciones desarrolladas en Adobe AIR deben tener una firma digital. Ésta se encarga de verificar nuestra identificación como desarrolladores a fin de brindarles mayor seguridad a los usuarios, es decir, informa respecto a si nuestras aplicaciones son confiables o no.

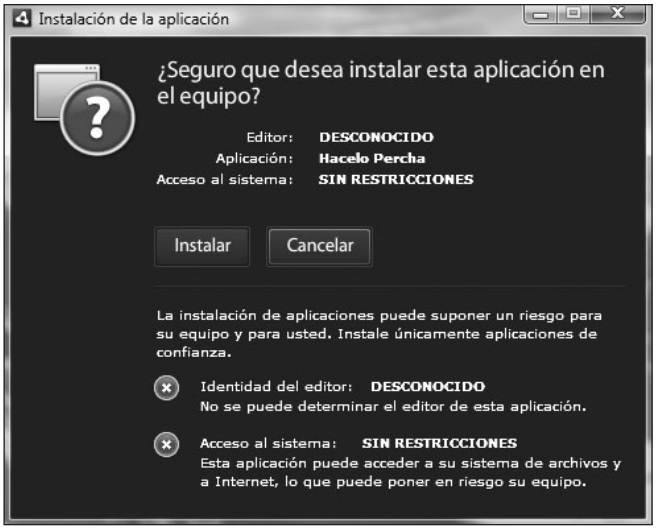


Figura 17. *Cuando instalamos una aplicación AIR y no poseemos un* ***certificado digital acreditado*** *por un organismo, se le informará al usuario por medio de una ventana.*

- **Destino:** desde esta opción podemos cambiar la ubicación donde se creará el archivo instalador. Por defecto, se situará en la misma carpeta en la que guardamos nuestro archivo .FLA.
- **Archivos incluidos:** en caso de utilizar algún archivo externo en nuestra aplicación, debemos incluirlo en esta lista de manera que, al publicar el instalador, éste se añada. En caso de haber utilizado un archivo MP3, JPG, FLV, SWF, etcétera, o si hicimos uso de una **base de datos**, debemos vincular estos archivos a nuestro desarrollo desde aquí.

Creación de una firma digital

En caso de no contar con una firma digital, debemos crearla. Mientras tanto, Flash permite generar instaladores sin firmar para que la aplicación se pueda validar más adelante. Entonces, tanto para crear una firma como para usarla, debemos presionar en el botón **Definir...** que se encuentra junto al texto que dice **Firma digital: Seleccionar certificado para firmar archivo de AIR**.

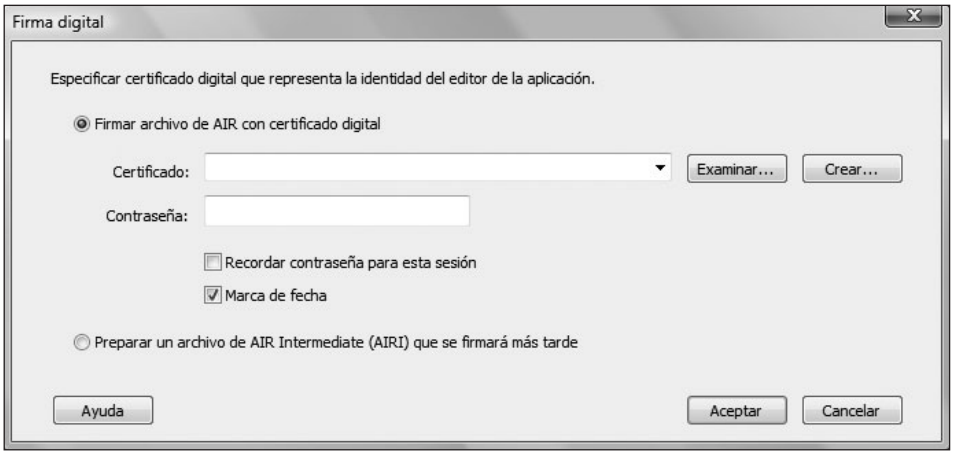


Figura 18. *Desde la ventana* ***Firma Digital****, podemos seleccionar una firma presionando el botón* ***Examinar...*** *o bien generar una haciendo clic en el botón* ***Crear...****.*

En caso de contar con una firma digital, debemos presionar el botón **Examinar...** y seleccionarla de nuestro sistema. En caso contrario, pulsamos el botón **Crear...**, que abrirá una nueva ventana en la que tendremos que definir los siguientes datos:

- **Nombre del editor**
- **Unidad de organización**
- **Nombre de organización**
- **País**
- **Contraseña**
- **Tipo**

Una vez que hemos definido esta información, presionamos el botón **Examinar...** a fin de establecer la ruta en la que guardaremos la firma digital. Veremos que sobre el campo de texto **Guardar como** aparecerá la ruta que acabamos de indicar. Finalmente, presionamos el botón **Aceptar** para concluir con la creación de nuestra firma digital. Lo único que resta es, en la ventana **Firma digital**, proceder a la carga de la firma, presionando el botón **Examinar...**.

III SQL LITE

Adobe Air nos permite hacer uso de bases de datos locales. **SQL lite** permite guardar información en pequeñas bases de datos compuestas por un solo archivo, lo que puede ser útil al momento de desarrollar aplicaciones de escritorio, como libretas de contactos o calendarios.

Figura 19. *En la ventana* ***Crear certificado digital con firma automática****, completamos los datos y creamos nuestra firma digital.*

Luego de realizar la configuración necesaria para nuestra publicación, presionamos **Aceptar** y habremos generado nuestra primera aplicación de escritorio.

DESARROLLO DE UN REPRODUCTOR DE MP3

Hasta el momento nos hemos familiarizado con este nuevo entorno de desarrollo y ya tenemos un importante panorama de las posibilidades que nos brinda Adobe AIR. A continuación, veremos de qué manera llevar a cabo el desarrollo de un reproductor de MP3 que sea capaz de reproducir archivos del sistema local luego de añadirlos a su lista de reproducción por medio de una ventana del sistema operativo o arrastrando los archivos de audio sobre nuestra aplicación.
Considerando que hemos destinado el **capítulo 6** de este libro al desarrollo de un reproductor de MP3 para la Web, no generaremos uno nuevo sino que reutilizaremos su interfaz gráfica. Lógicamente, habrá algunas variaciones:

- Cuando creamos nuestro reproductor de MP3, lo desarrollamos teniendo en mente que se trataba de una aplicación web, por lo que la carga de los contenidos se hizo mediante una estructura **XML** donde se encontraba la información de nuestros archivos de sonido para su posterior carga y reproducción. Esto no hará falta en nuestro ejemplo, ya que los sonidos los tomaremos del sistema.

- También desarrollamos una serie de funcionalidades que solamente son útiles al tratarse de aplicaciones web y que no necesitaremos al crear una aplicación local. Por ejemplo, el almacenamiento en **buffer**, ya que no hay tiempo de descarga en una aplicación que se ejecuta y trabaja de forma local.

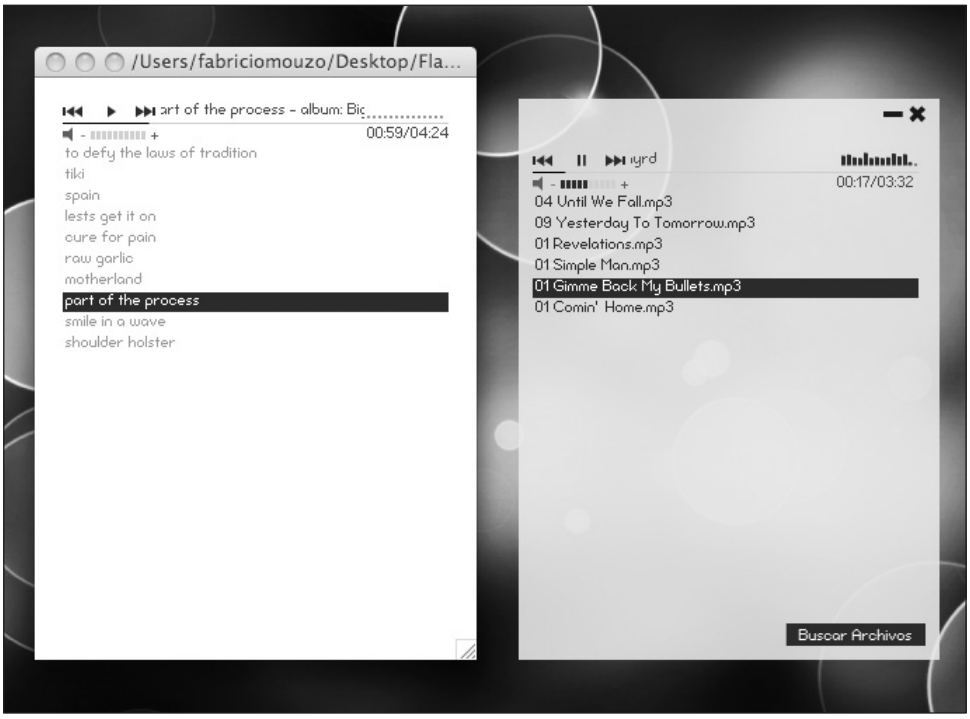


***Figura 20.** Nuestro reproductor para web llevado a cabo en el **capítulo 6** y nuestro próximo reproductor para escritorio. Veremos que son dos modos completamente distintos de pensar una aplicación.*

Si hay algo que no debemos hacer es caer en la comparativa entre un reproductor pensado para la Web y uno pensado como aplicación de escritorio: ambos poseen ventajas y desventajas, y es importante entender que **Flash Player** y **Adobe Integrated Runtime** tienen dos finalidades completamente distintas. Uno es óptimo para la Web y el otro es óptimo para aplicaciones de escritorio. La ventaja que tenemos es que manejando con fluidez ActionScript 3.0, podemos abarcar ambas ramas del desarrollo sin mayores problemas, y adaptar nuestras aplicaciones al entorno que sea conveniente para nuestra aplicación (o a ambos).

Desarrollo del proyecto

Teniendo en cuenta que utilizaremos la interfaz gráfica del reproductor de MP3 que creamos a lo largo del **capítulo 6**, no abarcaremos su desarrollo. Simplemente,

debemos utilizar los mismos contenidos visuales que en nuestro reproductor, y aquellos que debamos añadir los conoceremos cuando corresponda. Por el momento, nos abocaremos al desarrollo de nuestra aplicación.
Lo primero que haremos será ingresar al panel de configuración de AIR y cambiar el **Estilo de ventana** de su valor por defecto a **Fondo cromático personalizado (transparente)**. A la clase llamada **ReproductorMp3** que generamos en el **capítulo 6** debemos agregarle las siguientes clases, propias de la API de AIR. Iremos viendo cada una de ellas en detalle a lo largo del desarrollo de este capítulo para comprender su funcionalidad y forma de utilización.

```
import flash.filesystem.File;
import flash.display.NativeWindow;
import flash.desktop.NativeDragManager;
import flash.desktop.NativeDragActions;
import flash.events.NativeDragEvent;
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;
import flash.net.FileFilter;
import flash.filesystem.FileStream;
import flash.filesystem.FileMode;
```

Ventanas nativas: clase NativeWindow

Adobe AIR posee un robusto sistema de soporte para la ventana nativa (sobre la que corre nuestra aplicación) que cubre todas las funcionalidades que deseemos añadir a nuestras aplicaciones de escritorio.

Métodos de la clase nativeWindow

Cuando decidimos generar aplicaciones que utilizan ventanas personalizadas, debemos crear las funciones que permitan al usuario minimizar, cerrar, ajustar su tamaño o arrastrar la ventana por el escritorio.

FONDO CROMÁTICO PERSONALIZADO (TRANSPARENTE)

Debemos tener en cuenta que cuando utilizamos el fondo en modo transparente, Flash elimina el fondo de la aplicación, por lo que deberemos dibujarlo, ya sea por código o desde la IDE de Flash. En caso contrario, nuestra aplicación no contará con un fondo. La ventaja de esto es que lo podemos crear nosotros mismos y generar un valor agregado en nuestra aplicación por medio de su diseño.

Para agregar estas funcionalidades a nuestro proyecto, usaremos los métodos propios de la clase **NativeWindow**, que detallamos en la siguiente tabla.

MÉTODO	DESCRIPCIÓN
nativeWindow.close()	Con este método cerraremos la ventana a la que hagamos referencia. Si contamos con diez ventanas, únicamente se cerrará la que contenga el método.
nativeWindow.minimize()	Permite al usuario minimizar la ventana.
nativeWindow.maximize()	Permite maximizar la ventana.
nativeWindow.startMove()	Permite arrastrar la aplicación por el escritorio.
nativeWindow.startResize()	Permite ajustar el tamaño de la ventana.
nativeApplication.exit()	A diferencia de **close()**, permite cerrar la aplicación entera. Si contamos con diez ventanas, todas se cerrarán.

***Tabla 2.** Métodos de la clase NativeWindow para manejar las ventanas de las aplicaciones.*

Sabiendo los métodos por medio de los cuales podemos dotar a nuestra aplicación de las funcionalidades básicas, a partir de ahora veremos de qué manera implementarlos en nuestro código.

Comportamientos para nuestra ventana nativa

Para hacer uso del método **close()**, simplemente debemos aplicarlo a la ventana nativa que queremos cerrar. En nuestro caso, sólo utilizaremos una única ventana, por lo que lo aplicaremos al botón que se encarga de cerrar nuestro reproductor de MP3, como vemos a continuación:

```
private function cerrarAplicacion(e:MouseEvent):void{
     stage.nativeWindow.close();
}
```

La implementación de los métodos **startMove()**, **maximize()** y **minimize()** es igual que la del método **close()**; simplemente debemos asignarle ese método al botón o clip que se encargará de ejecutar cada uno de ellos:

```
private function minimizarAplicacion(e:MouseEvent):void {
     stage.nativeWindow.minimize();
}
private function moverAplicacion(e:MouseEvent):void {
     stage.nativeWindow.startMove();
}
```

Figura 21. *Al personalizar nuestra ventana, debemos programar los comportamientos necesarios para* ***cerrar*** *y* ***minimizar*** *la aplicación. Para ello, utilizamos los métodos* ***close()*** *y* ***minimize()****.*

Sistema local de archivos

Hasta el momento, hemos visto los comportamientos para crear la interfaz gráfica de nuestra aplicación, es decir, las diversas funcionalidades de nuestra ventana nativa para poder cerrarla, minimizarla o moverla sobre el escritorio. A continuación, conoceremos cómo hacer interactuar nuestra aplicación con los archivos del sistema local. Aquí es donde en verdad comienza a ser interesante el uso de Adobe AIR, y donde podremos conocer su máximo potencial.

Es importante mencionar que existe una gran diferencia entre una aplicación AIR de escritorio y un archivo SWF convencional. Si bien un archivo SWF nos permite acceder a determinada información de archivos del sistema, por medio de las clases **FileReference** y **FileReferenceList**, y realizar algunas operaciones con ellos (enviarlos a un lenguaje back-end para almacenarlos, por ejemplo), con una apli-

MÉTODO ALWAYSINFRONT

Podemos definir que nuestra aplicación nunca sea tapada por otro programa del sistema operativo, de manera tal que siempre se vea por sobre las demás aplicaciones. Para ello debemos utilizar la propiedad **alwaysInFront**, de la clase **NativeWindow**, asignándole el valor **true**.

cación desarrollada en Adobe AIR las posibilidades son mucho mayores: no sólo podemos leer esos archivos o enviarlos a un servidor sino que podemos **copiarlos**, **modificarlos** y **borrarlos**; es decir, tenemos un acceso mucho más ilimitado, lo que nos brinda una enorme gama de posibilidades a las que no tenemos acceso desde una película SWF convencional.

Añadir archivos locales al reproductor

Permitir al usuario seleccionar archivos del sistema local no es una tarea compleja, si comprendemos las clases que están involucradas en este proceso y las tareas que realizan dentro de él. Como sería muy engorroso que los usuarios debieran seleccionar de a uno los archivos MP3 que desean escuchar, para simplificar este proceso brindaremos la posibilidad de añadir a la lista de reproducción todos los archivos simultáneamente. Realizar esta tarea es posible a través de la utilización de la clase **FileListEvent**, que permite seleccionar múltiples archivos y a su vez genera un listado que contiene toda la información de los archivos que hemos elegido. En primer lugar, debemos importar la clase:

```
import flash.events.FileListEvent;
```

Luego, tenemos que crear el botón que permitirá generar la búsqueda de los archivos a través de la composición de una función que llamaremos **buscarArchivos()**.

```
buscarBtn = new BtnBuscar();
buscarBtn.x=stage.stageWidth-buscarBtn.width-10;
buscarBtn.y=stage.stageHeight-buscarBtn.height-10;
buscarBtn.buttonMode=true;
buscarBtn.addEventListener(MouseEvent.CLICK, buscarArchivos,false,0,true);
addChild(buscarBtn);
```

Veamos la función **buscarArchivos()**, que es la que realmente nos interesa:

MÉTODO STARTMOVE()

Cuando utilizamos el método **startMove()** para arrastrar una ventana sobre el escritorio, no es necesario asignar un listener para saber cuándo el usuario libera nuestra aplicación. Adobe AIR detecta automáticamente cuándo esto sucede, simplificándonos la tarea a los desarrolladores, que nos ahorramos el empleo del listener para reconocer esa acción del usuario.

```
private function buscarArchivos(e:MouseEvent):void {
     archivo = new File();
     archivo.browseForOpenMultiple("Seleccione archivos mp3 o
wav",[filtroDeArchivos]);
     archivo.addEventListener(FileListEvent.SELECT_MULTIPLE,
seleccionarArchivos, false, 0, true);
}
```

Primero crearemos una instancia de la clase **File**, a la que llamaremos **archivo**. Una vez que contamos con ella, le añadimos el método **browseForOpenMultiple**, que se encarga de abrir un cuadro de diálogo del sistema para que el usuario pueda realizar la búsqueda de los archivos y seleccionarlos. Es importante conocer sus dos parámetros:

- **título**: debemos indicar la cadena de texto que aparecerá en el cuadro de diálogo; en nuestro caso, ingresamos el texto **"Seleccione archivos mp3 o wav"**.
- **tipoFiltro**: en este parámetro podemos determinar una instancia de la clase **FileFilter**, que nos permite filtrar los archivos que se mostrarán en el cuadro de diálogo. En caso de omitir este parámetro, se mostrarán todos los archivos del sistema operativo. Notemos que a este segundo parámetro, le asignamos la variable **filtroDeArchivos**:

```
 private var filtroDeArchivos:FileFilter=new FileFilter("Archivos de
Audio","*.mp3;*.wav;");
```

En verdad, la variable **filtroDeArchivos** es una instancia de la clase **FileFilter**, por medio de la cual filtramos los archivos que se mostrarán en el cuadro de diálogo. Si nuestro desarrollo es un reproductor de audio, no tiene ningún sentido que el usuario tenga que estar buscando una canción entre imágenes, videos y cuanto material tenga en su sistema operativo. De este modo, sólo se mostrarán en el cuadro de diálogo sus archivos de audio, sean estos MP3 o WAV. Podemos asignar cuantas extensiones necesitemos, este sería un ejemplo de cómo filtraríamos únicamente imágenes:

PROPIEDADES Y MÉTODOS DE LA CLASE FILE

En el sitio **http://help.adobe.com/es_ES/AS3LCR/Flash_10.0/flash/filesystem/File.html** encontraremos más información respecto al uso de la clase **File**, las distintas alternativas que nos da y su alcance en el desarrollo de aplicaciones de escritorio. Este capítulo es tan sólo un ejemplo de las posibilidades que nos brinda Adobe AIR por medio de la aplicación de esta clase.

```
private var filtroDeArchivos:FileFilter=new FileFilter("mostrar algunas
imágenes","*.jpg;*.bmp; *.tif; *.png; *.gif;");
```

En este caso, nuestro cuadro de diálogo solamente mostraría imágenes cuyas extensiones sean JPG, BMP, TIF, PNG o GIF.

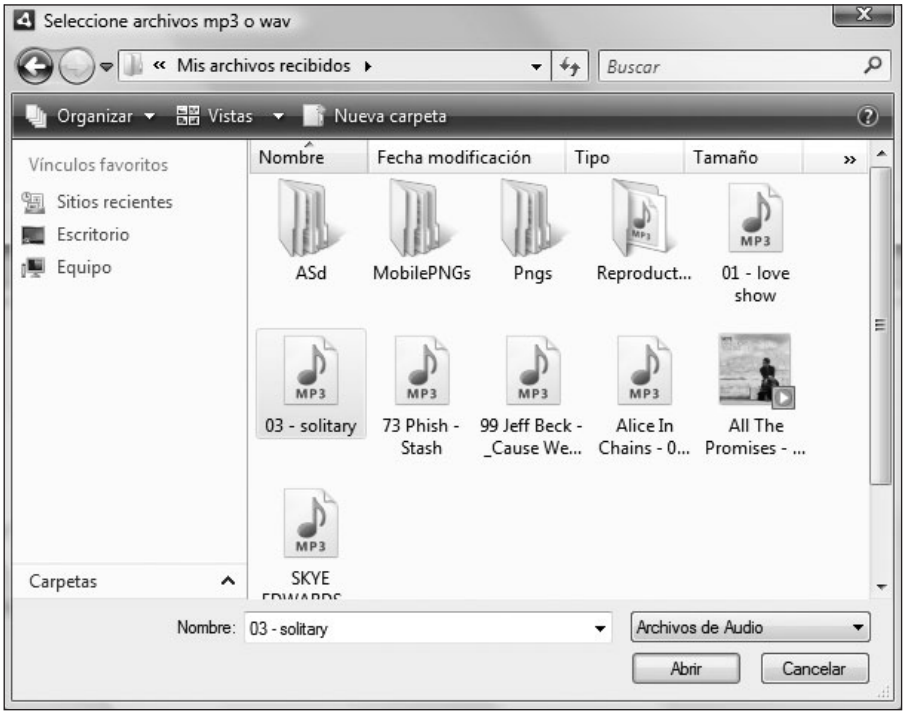


Figura 22. *Gracias a la clase **FileFilter**, se mostrarán solamente archivos de audio **MP3** o **WAV**. También, veremos que al lado de **Tipo**, aparece la cadena de texto que hemos definido: **Archivos de Audio**.*

Finalmente, añadimos un listener a nuestra instancia para detectar cuándo el usuario ha concluido la selección de los archivos, como vemos a continuación:

FILTRO DE ARCHIVOS

Si no utilizamos la clase **FileFilter**, es probable que cuando el usuario seleccione los archivos agregue alguno con una extensión inapropiada para nuestra aplicación y esto ocasione que ésta falle o deje de funcionar. Es por esto que debemos tener en cuenta qué tipo de archivos serán permitidos para nuestra aplicación y sus respectivas extensiones.

```
archivo.addEventListener(FileListEvent.SELECT_MULTIPLE, seleccionarArchivos,
false, 0, true);
```

Cuando detectamos que el usuario terminó de seleccionar archivos, llamamos a la función **seleccionarArchivos()**, que veremos a continuación:

```
private function seleccionarArchivos(event:FileListEvent):void{
 for (var m:uint = 0; m < event.files.length; m++){
     var ruta:String= event.files[m].url;
     var nombre:String= event.files[m].name;
     var songObject:Object={ruta:ruta,nombre:nombre};
     songObject.id=currentId;
     temasArray.push(songObject);
     cargarTema(temasArray[currentId].id);
     actualizarPlaylist();
     currentId++;
     }
}
```

Ya utilizamos esta sintaxis a lo largo de este libro. En este caso, como los usuarios pueden seleccionar uno o varios archivos a la vez, necesitamos de un **operador** que nos permita recorrer la totalidad de los archivos que han sido seleccionados. Podemos conocer este valor accediendo a la propiedad **length** del array **files**:

```
for (var m:uint = 0; m < event.files.length; m++) {
```

De este modo, el bucle **for** se ejecutará tantas veces como archivos hayamos seleccionado. Luego, dos líneas cambian respecto a nuestro desarrollo en el **capítulo 6**:

```
var ruta:String= event.files[m].url;
var nombre:String= event.files[m].name;
```

Si bien en el **capítulo 6** debíamos tomar la información de una estructura **XML**, en este caso debemos tomarla de los archivos que acabamos de agregar a nuestra aplicación. Desde el array **files** que incluye el evento, accedemos a cada posición de éste (variable **m**), a su **ruta** y a su **nombre** por medio de las propiedades **url** y **name**. Finalmente, creamos un objeto con esta información por cada archivo que se haya agregado.

```
var songObject:Object={ruta:ruta,nombre:nombre};
```

Luego, almacenamos ese objeto dentro de un array:

```
temasArray.push(songObject);
```

Arrastrar contenidos a la aplicación

La mayoría de las aplicaciones de escritorio permiten arrastrar archivos sobre ellas para luego utilizarlos. Veremos de qué manera podemos llevar esto a cabo en Adobe AIR a fin de facilitarle al usuario la posibilidad de agregar temas arrastrándolos y soltándolos sobre la aplicación. Para poder dotar de esta funcionalidad a nuestro reproductor, debemos definir qué objeto actuará como **receptor** dentro de la aplicación. En nuestro reproductor, estableceremos como receptor al **MovieClip backgroundMC** de nuestra interfaz, el cual simula el fondo de nuestra aplicación. Al receptor debemos asignarle dos eventos propios de la clase **NativeDragevent**, que servirán para que la aplicación sepa cuándo un archivo es arrastrado y cuándo se ha soltado:

```
backgroundMc.addEventListener(NativeDragEvent.NATIVE_DRAG_ENTER, enterDrag,
false, 0, true);

backgroundMc.addEventListener(NativeDragEvent.NATIVE_DRAG_DROP, dropDrag,
false, 0, true);
```

En la función **enterDrag()** vamos a crear un contenedor por medio de la clase **Clipboard**, al que llamaremos **escritorio**, que nos permitirá transferir los datos de los archivos a la aplicación. Esto nos sirve para que la aplicación conozca desde dónde provienen los archivos que los usuarios arrastrarán hasta ella, es decir, cuál es la ruta del archivo en nuestro sistema operativo:

```
private function enterDrag(e:NativeDragEvent):void {
     var escritorio:Clipboard=e.clipboard;
```

Luego, debemos implementar un **condicional** para saber si los elementos que se están arrastrando sobre nuestra aplicación son realmente archivos reconocibles por Adobe AIR y también tenemos que especificar de qué manera queremos recibir esos archivos. Esto lo averiguamos por medio del método **hasFormat()** de la clase **Clipboard**. Este mé-

todo nos permite comprobar si existen datos de formato en los archivos que se hayan arrastrado y de qué manera debemos obtener la información. Debido a que necesitamos tener acceso a un listado de archivos y a sus rutas, empleamos **FILE_LIST_FORMAT** para recibir la información de los archivos dentro de un array:

```
if(escritorio.hasFormat(ClipboardFormats.FILE_LIST_FORMAT)) {
     NativeDragManager.acceptDragDrop(backgroundMc);
}
```

En caso de que se dé esta condición, tenemos que crear una instancia de la clase **NativeDragManager** y aplicarle el método **acceptDragDrop()**, asignándole como parámetro nuestro clip receptor: **backgroundMc**. Este método ejecutará la función **dropDrag()**, dentro de la cual ya podemos manipular los archivos en nuestra aplicación:

```
private function dropDrag(e:NativeDragEvent):void {
var archivosDragArray:Array=e.clipboard.getData(ClipboardFormats.
FILE_ LIST_FORMAT) as Array;
```

En primer lugar, lo que hacemos es transferir la información a un array al que denominamos **archivosDragArray**. A partir de aquí, simplemente debemos manipular los archivos que contiene este array de manera muy similar al modo en el que lo hicimos cuando creamos la función para la búsqueda de archivos del sistema local por medio del cuadro de diálogo.
Una vez que tenemos la lista de archivos convertida en un array, debemos generar un bucle **for** para manipular cada uno de estos archivos de la misma manera en que lo hicimos anteriormente al crear la función **seleccionarArchivos()**.

```
for (var m:uint; m<archivosDragArray.length; m++) {
     var archivoDrageado:File=archivosDragArray[m] as File;
}
if (archivoDrageado.extension==”mp3”) {
     for each (archivoDrageado in archivosDragArray) {
     var ruta:String=archivoDrageado.url;
     var nombre:String=archivoDrageado.name;
     var songObject:Object={ruta:ruta,nombre:nombre};
     songObject.id=currentId;
     temasArray.push(songObject);
     cargarTema(temasArray[currentId].id);
```

```
        actualizarPlaylist();
        currentId++;
        }
}
```

Sólo resta publicar nuestro reproductor del mismo modo en que lo hicimos con el navegador web que generamos al principio del capítulo. Páginas atrás tenemos todas las opciones posibles de publicación.

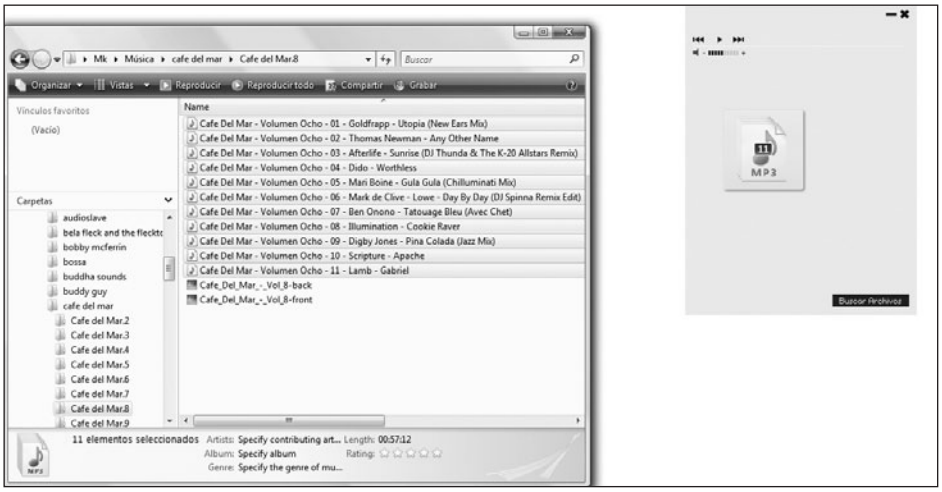


Figura 23. *Tan sólo arrastrando los archivos de audio sobre nuestra aplicación, ésta los tomará y los agregará a la* ***lista de reproducción****.*

RESUMEN

A lo largo de este libro hemos aprendido a utilizar la API de dibujo de Flash, formularios, imágenes, sonidos, videos, webcams, micrófonos, entre otras tantas cosas. La intención de este capítulo es brindar una herramienta que nos permita potenciar todo aquello que estudiamos anteriormente y, con un pequeño proceso de análisis, poder llevar al escritorio de los usuarios la posibilidad de desarrollo de RIAs. Adobe AIR potencia Flash y nos ofrece volcar al escritorio del usuario, todo aquel potencial que nos otorga Flash como plataforma. Las posibilidades son inimaginables.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Cuáles son las principales diferencias entre una aplicación web y una de escritorio?

2. ¿En qué potencia Adobe AIR a Flash?

3. ¿Cómo creamos un archivo AIR?

4. ¿Cómo exportamos un proyecto en AIR?

5. ¿Qué tipos de ventanas nativas podemos utilizar en un desarrollo en AIR y cuáles son sus diferencias?

6. ¿Qué diferencia hay entre el método **close()** y el método **exit()** de **NativeWindow**?

7. ¿Con qué método minimizamos una pantalla? ¿Con cuál la maximizamos?

8. ¿Para qué sirve el método **startMove()**?

9. ¿Para qué se utiliza la clase **FileFilter**?

10. ¿Qué clases se ven involucradas al momento de arrastrar archivos a una aplicación?

EJERCICIOS PRÁCTICOS

1. Modifique la galería de fotos del capítulo de imágenes para que utilice archivos del sistema local.

2. Genere, con Adobe Air, un reproductor de videos FLV que reproduzca videos que se encuentren en el sistema local.

3. Desarrolle una aplicación AIR que permita dibujar sobre el escritorio.

4. Genere una aplicación AIR que permita arrastrarle fotos para visualizarlas y dibujar sobre ellas.

5. Cree una aplicación experimental en Adobe AIR que involucre el uso de imágenes a través de una API de cualquier servicio web (flickr, Picasa, etcétera).

Manejo de archivos

A lo largo de este libro hemos aprendido de qué manera enviar y recibir variables, cómo enviar imágenes y cargar contenidos dinámicos, pero no hemos explicado de qué forma subir archivos desde Flash. Por medio de las clases FileReference y FileReferenceList podemos enviar uno o varios archivos a un servidor. A continuación, veremos de qué modo hacerlo.

SUBIR Y DESCARGAR ARCHIVOS

Aunque las clases **FileReference** y **FileReferenceList** no son una novedad de ActionScript 3.0, vale la pena dedicarles un apéndice, en especial considerando los temas que hemos visto. Incluso, si bien no son clases nuevas para la plataforma, la sintaxis que AS3 propone hace mucho más sencillo e intuitivo su uso.
Estas clases nos permiten subir archivos desde Flash a un servidor o descargar archivos desde un servidor a Flash. Con anterioridad a su existencia, era muy complejo lograr este procedimiento, ya que había que utilizar JavaScript como puente entre el cliente (Flash) y el servidor. Actualmente, gracias a estas clases, el procedimiento es completamente transparente y de fácil implementación.

Diferencias entre FileReference y FileReferenceList

Si bien la mayoría de los métodos y propiedades de ambas clases son prácticamente los mismos, la diferencia radica en que la clase **FileReference** referencia a un único archivo del sistema operativo del usuario, mientras que la clase **FileReferenceList** nos permite acceder a varios archivos. El hecho de que la clase **FileReferenceList** posibilite trabajar con varios archivos al mismo tiempo no implica que sea más útil que la clase **FileReference**; por el contrario, cada una brinda una mejor performance ante determinados casos en particular.
Imaginemos un sencillo ejemplo: una web nos facilita subir videos a un servidor. Para ello, debemos acceder a nuestro sistema por medio de una ventana de diálogo y seleccionarlo. No tendría ningún sentido emplear la clase **FileReferenceList** en este caso ya que el usuario debe seleccionar un único archivo. Todo lo contrario sucede con un administrador de imágenes. Sería por demás molesto que el usuario tuviera que subir una por una cada una de sus imágenes. En este caso, por medio de la clase **FileReferenceList**, le facilitamos al usuario la posibilidad de que seleccione todas las imágenes que sean necesarias. Son los desarrolladores quienes deben saber cuándo emplear una clase y cuándo otra. Si bien en el siguiente ejemplo haremos uso de la clase **FileReference**, todos sus métodos y propiedades son aplicables, también, a la clase **FileReferenceList**.

III INJERENCIA DEL MANEJO DE ARCHIVOS EN LA WEB 2.0

Si bien las clases tratadas en este apéndice han sido siempre de gran utilidad, es indudable que su importancia fue creciendo conforme al desarrollo de la Web. En sus comienzos, difícilmente se subían archivos desde Flash, pero con la aparición de la Web 2.0 y el rol activo del usuario como creador de contenidos, estas clases se emplean cada vez con mayor frecuencia.

Abrir el cuadro de diálogo del sistema operativo

Antes de comenzar a ver los métodos y propiedades más útiles de esta clase, debemos generar una instancia de ella, como vemos en la siguiente línea:

```
var miFileReference:FileReference = new FileReference();
```

Una vez que contamos con una instancia de la clase, podemos acceder a sus métodos y propiedades. En primer lugar, haremos uso del método **browse()**, que es el encargado de abrir el cuadro de diálogo del sistema operativo.

```
miFileReference.browse()
```

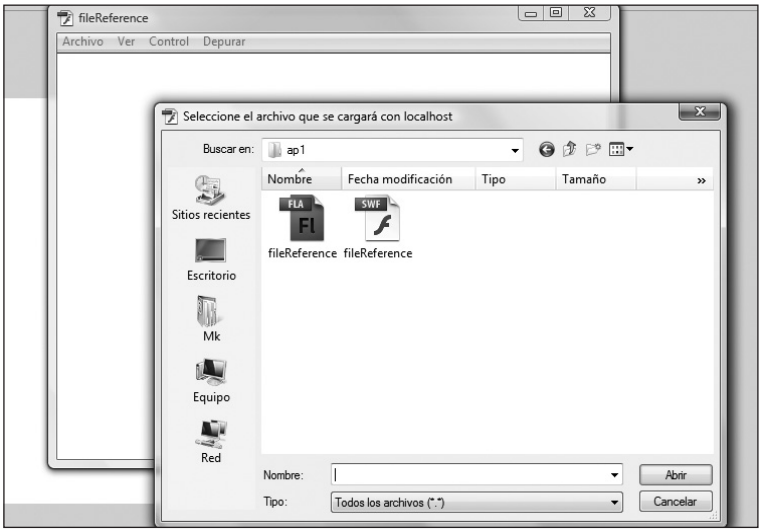


Figura 1. *Por medio del método browse() abrimos el cuadro de diálogo del sistema operativo. Con FileReference, podemos seleccionar un único archivo. Con FileReferenceList, podemos seleccionar varios.*

Definir tipos de archivos disponibles

Por medio de la clase **FileFilter** podemos definir qué tipos de archivos permitiremos que sean visibles dentro del cuadro de diálogo y el texto que acompañará a ese cuadro. La enorme ventaja de emplear esta clase es que le simplificamos al usuario la tarea de buscar los archivos dentro de su equipo. Suponiendo que desarrollemos un administrador que permita al usuario subir **imágenes** y **sonidos**, utilizamos la clase **FileFilter** y filtramos para que el usuario pueda visualizar archivos con las extensiones más usuales de estos dos tipos de archivos:

```
var filtroImagenes:FileFilter = new FileFilter("seleccionar imagenes",
"*.jpg;*.gif;*.png");
var filtroSonidos:FileFilter = new FileFilter("seleccionar sonidos",
"*.mp3;*.wav");
```

Una vez creadas nuestras dos instancias de la clase **FileFilter**, las aplicamos como parámetros al método **browse()** visto anteriormente. Dentro de la matriz que debemos indicar como parámetro, podemos asignar la cantidad de instancias de la clase **FileFilter** que consideremos necesarias.

```
miFileReference.browse([filtroImagenes, filtroSonidos]);
```

Figura 2. *Una vez aplicado el filtro, sólo aparecerán los archivos cuyas extensiones hayamos definido. Prestemos especial atención al menú desplegable que se encuentra al lado de* ***Tipo****.*

Eventos para el manejo de los contenidos

Abrir el cuadro de diálogo es sólo el comienzo para el manejo de contenidos. A partir de aquí, el usuario tiene dos alternativas: seleccionar un archivo (o varios en caso de emplear la clase **FileReferenceList**) o cerrar el cuadro de diálogo. Para detectar estos sucesos, ActionScript pone a nuestra disposición los eventos **cancel** y **select**, que nos informan al respecto. Veamos cómo implementarlos.

Evento cancel

Para ver en funcionamiento a estos eventos, crearemos dos campos de texto dinámicos. A uno de ellos lo llamaremos **browse_txt** y lo utilizaremos para indicar el

archivo que ha sido seleccionado; y al otro lo llamaremos **info_txt** y lo emplearemos para indicar qué operación se ha realizado. Para saber si el usuario canceló el cuadro de diálogo, simplemente asignamos un listener a nuestra instancia de la clase **FileReference** y creamos una función para él. Si éste se ejecuta, es porque el usuario cerró la ventana de diálogo. En este caso, lo indicamos en el campo de texto **info_txt**. Veamos, a continuación, el código que debemos utilizar para llevar a cabo estas operaciones:

```
var miFileReference:FileReference = new FileReference();
miFileReference.addEventListener(Event.CANCEL, cancelarSeleccion, false, 0,
true);
function cancelarSeleccion(event:Event):void{
     info_txt.text = "cerraste el cuadro de dialogo!";
}
```

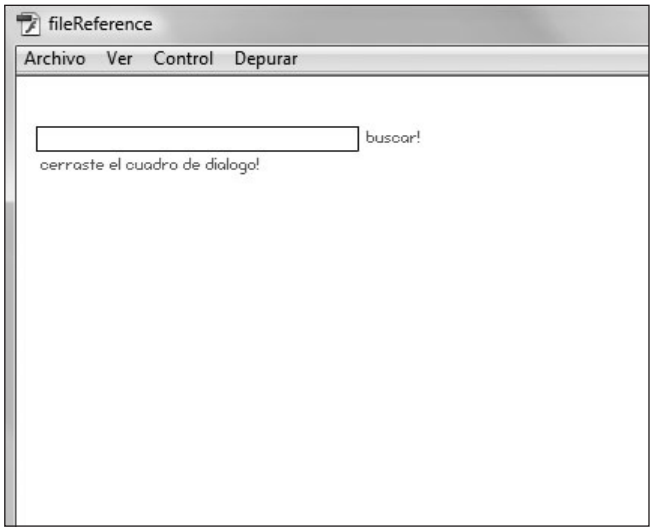


***Figura 3.** El evento cancel se ejecuta en respuesta al cierre del cuadro de diálogo del sistema operativo.*

III ALTAS, BAJAS Y MODIFICACIONES DESDE FLASH

Solía ser difícil imaginar un **ABM** (**Altas, Bajas, Modificaciones**) desarrollado en Flash, pero con los **métodos** y **propiedades** de las clases **FileReference** y **FileReferenceList**, podemos crearlo sin ningún inconveniente. Al ser un tema nuevo, el mejor modo de familiarizarse con estos conceptos es observando las características de estos sistemas, tomar sus buenas prácticas e incorporarlas.

Evento select

Si bien el evento **cancel** nos puede ser de ayuda, el evento que nos interesa **escuchar** es **select**, que se ejecuta cuando el usuario selecciona uno o varios archivos de su sistema operativo y presiona el botón **Abrir** dentro de la interfaz del cuadro de diálogo. A partir de aquí, tenemos a nuestra disposición una serie de propiedades que veremos a continuación. En primer lugar, asignamos el listener del mismo modo que lo hicimos para el evento **cancel** y luego creamos su respectiva función:

```
miFileReference.addEventListener(Event.SELECT, seleccionarArchivo, false, 0,
true);
function seleccionarArchivo(event:Event):void{
     browse_txt.text = miFileReference.name;
}
```

Propiedades de FileReference y FileReferenceList

Las clases **FileReference** y **FileReferenceList** cuentan con una serie de propiedades que están disponibles, por razones lógicas, una vez que el usuario selecciona uno o varios archivos. Conozcamos cuáles son.

Propiedad name

La propiedad **name** nos informa respecto al nombre del archivo que ha seleccionado el usuario. Generalmente, lo más común es introducir en un campo de texto el nombre del archivo que ha abierto el usuario. Esto es en verdad útil suponiendo que el usuario haya cometido un error: informándole respecto al nombre, puede eliminar el archivo o seleccionar otro en su reemplazo.

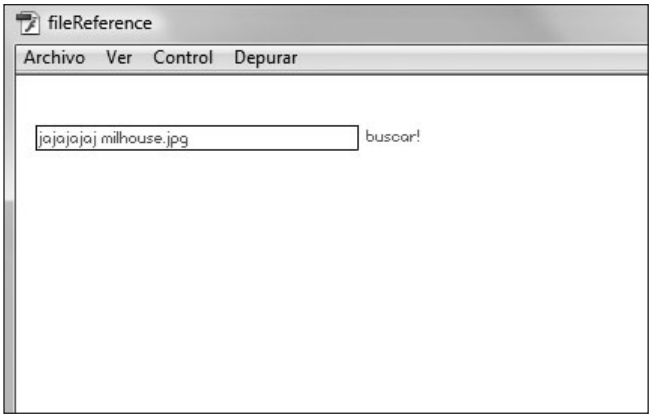


Figura 4. *La propiedad* ***name*** *está disponible una vez que el usuario selecciona un archivo. Indicamos su valor en el campo de texto llamado* ***browse_txt.***

Propiedad size

La propiedad **size** es de gran importancia. Por medio de ella, accedemos al peso del archivo que el usuario ha seleccionado. Esto es realmente útil para los casos en los que en nuestro sistema queremos restringir el **peso** de los archivos que se van a utilizar. Este procedimiento, generalmente, lo llevan a cabo todos los sistemas de archivos. De este modo se evitan errores, se restringe el máximo permitido y tenemos un control mucho más exacto sobre el servidor y su espacio. A su vez, eliminamos las posibilidades de error por parte del usuario y limitamos los riesgos de colapsar el servidor en caso de que un usuario mal intencionado deje subiendo grandes cantidades de contenidos.

Debemos saber que el valor devuelto por la propiedad **size** está expresado en **bytes**. Supongamos que en nuestro sistema, permitimos solamente la carga de archivos que no superen los 4 **Mb** de tamaño. Sabemos que 1024 bytes equivalen a 1 kilobyte y que 1024 kilobytes componen 1 megabyte. Conociendo esta información básica, podemos agregar un condicional al seleccionar el archivo y así definir si el archivo es permitido o sobrepasa el máximo establecido:

```
function seleccionarArchivo(event:Event):void{
     if(miFileReference.size>(1024 * 1024 * 4)){
            info_txt.text = "el archivo excede el máximo permitido!";
     }else{
            browse_txt.text = miFileReference.name;
            trace(miFileReference.size);
     }
}
```

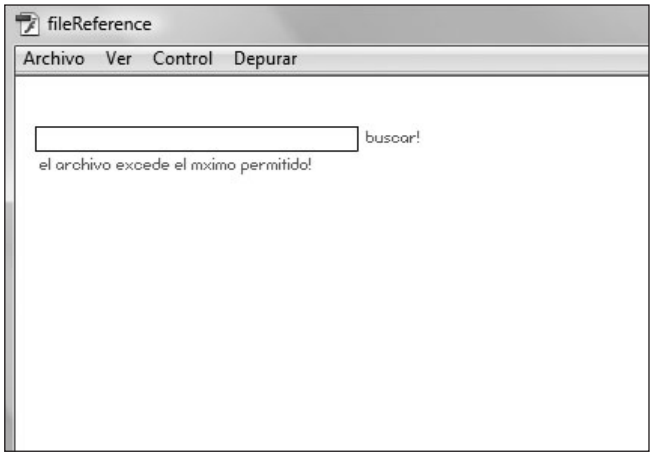


Figura 5. *En caso de que el peso del archivo sea mayor al máximo permitido, alertamos al usuario a través del campo de texto info_txt.*

Subir un archivo al servidor

Una vez que el usuario seleccionó un archivo y éste cumple con las condiciones que especificamos respecto a su tamaño (recordemos que como máximo puede pesar 4 Mb), debemos enviarlo al servidor. Para lograrlo, empleamos el método **upload()** de la clase **FileReference**. A ese método, debemos indicarle como parámetro una instancia de la clase **URLRequest** que contenga el nombre del archivo del lado del servidor al cual enviaremos el archivo seleccionado por parte del usuario. Dentro del condicional que creamos anteriormente, agregamos las siguientes líneas en caso de que se dé la condición. De este modo, se iniciará la carga del archivo al servidor. Veamos:

```
var miURLRequest:URLRequest = new URLRequest();
miURLRequest.url = "http://localhost/ejemploUpload/uploadImage.php";
miFileReference.upload(miURLRequest);
```

El archivo **uploadFile.php** será el encargado de recibir el archivo que estamos enviando desde Flash. Sabemos que no es PHP nuestro interés en este libro y por este motivo, limitamos su código a lo mínimo indispensable para lograr el objetivo propuesto en nuestro ejemplo:

```
<?php
$path = 'archivos/' . $_FILES['Filedata']['name'];
     if(move_uploaded_file($_FILES['Filedata']['tmp_name'], $path)){
     }
?>
```

En primer lugar, debemos crear una variable a la cual llamaremos **$path** y luego, le concatenamos el nombre del directorio en el que queremos almacenar los archivos que subamos (**archivos/**) y el nombre del archivo que acabamos de subir (**$_FILES['Filedata']['name']**). Por medio del método **move_uploaded_file()**, mo-

III OTRAS PROPIEDADES DE LA CLASE FILEREFERENCE

Si bien no abarcaremos otras propiedades en este apéndice, hay dos de ellas que nos pueden ser de utilidad. Por medio de la propiedad **type** podemos acceder al tipo de archivo y por medio de la propiedad **modificationDate** es posible averiguar la última fecha de modificación del archivo seleccionado dentro del disco del usuario.

vemos el archivo a la ruta que indicamos en la variable **$path**. Lógicamente, en un sistema profesional, interviene una cantidad considerable de código adicional por medio del que asignamos nombres con identificadores únicos a los archivos y almacenamos información en bases de datos, entre otras cosas. Lo único que aquí nos interesa es ver la sencillez con la que podemos mover un archivo del sistema operativo del usuario a un servidor.

***Figura 6.** Los archivos subidos al servidor se almacenarán dentro de la carpeta archivos/.*

Una de las grandes ventajas de las propiedades y métodos de la clase **FileReference** es que nos permite tener un control mucho más estricto sobre el proceso que se está llevando a cabo del lado del cliente, de manera tal que al momento de enviar los archivos al servidor, ya sabemos que:

LENGUAJE DEL LADO DEL SERVIDOR: PHP

Al igual que en capítulos anteriores, necesitamos de un lenguaje del lado del servidor que reciba nuestros archivos para luego manipularlos. En nuestros ejemplos, elegimos PHP. Recordemos que debemos probar estos ejemplos en un servidor con soporte para PHP o contar con un servidor local para verlos en funcionamiento.

- La extensión del archivo es válida.
- El peso del archivo es menor o igual al máximo permitido.

Esto nos posibilita generar administradores de contenido mucho más agiles y establecer una comunicación más transparente entre cliente y servidor, donde enviamos archivos solamente en caso de que cumplan con los requisitos especificados.

Eventos para el control de carga

Si bien hemos visto que contamos con los eventos **select** y **cancel**, podemos considerarlos previos al proceso de subida de un archivo. También disponemos de eventos que nos permiten controlar el proceso de carga, cuándo esta finalizó y si fue exitosa. Veamos los eventos que están involucrados en este proceso.

Progreso de la carga: evento progress

El evento **progress** es de gran utilidad, ya que por medio de él podemos acceder a información que nos permite conocer en qué porcentaje se ha producido la carga o la descarga de un archivo. Conociendo esta información, a través de una regla de tres simple, le informamos al usuario respecto al progreso del proceso, ya sea por medio de un campo de texto, o bien escalando un **Sprite** o **MovieClip**:

```
miFileReference.addEventListener(ProgressEvent.PROGRESS, progreso, false, 0,
true);
function progreso(event:ProgressEvent):void{
     info_txt.text = ((event.bytesLoaded / event.bytesTotal)*
100).toString() + " % cargado";
}
```

Accediendo a las propiedades **bytesLoaded** y **bytesTotal** del evento **progress**, establecemos una regla de tres y aplicamos su valor al campo de texto **info_txt**. De este modo, sabemos qué porcentaje de la imagen ha sido subida al servidor.

III USO DE ADMINISTRADORES DE CONTENIDOS EN LA WEB 2.0

Una buena práctica es observar los comportamientos de los administradores de contenidos de los sitios más respetados en lo que a Web 2.0 respecta. Esto nos ayuda a comprender mejor el porqué de cada procedimiento y a optimizar todo para brindarle más solvencia, seguridad y transparencia al usuario. El administrador de contenidos de **Flickr** es un gran ejemplo para tener en cuenta.

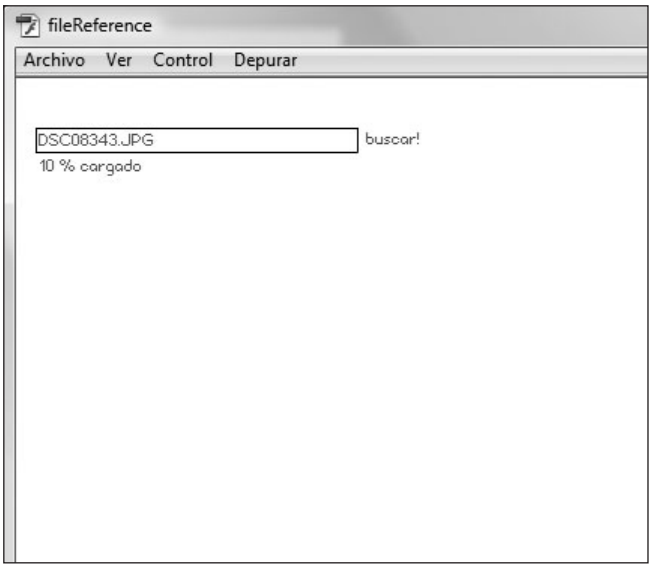


Figura 7. *Empleamos el campo de texto info_txt para informarle al usuario el porcentaje del archivo que ha sido enviado al servidor.*

Carga completa: evento complete

Por medio del evento **complete**, podemos saber cuándo ha concluido la carga de un archivo. Al ejecutarse este evento, sabremos que la carga ha sido exitosa y que el archivo se encuentra en el servidor.

```
miFileReference.addEventListener(Event.COMPLETE,cargaCompleta, false, 0,
true);
function cargaCompleta(event:Event):void{
     info_txt.text = “carga completa! :)”;
}
```

Este evento es de utilidad en especial al usar la clase **FileReferenceList**, ya que necesitamos saber cuándo se ha concluido el envío de un archivo para proceder con el siguiente.

FALLA EN CARGA DE ARCHIVO

Si bien el evento **complete** nos informa cuándo un archivo ha sido subido exitosamente al servidor, también contamos con un método que nos avisa en caso de que haya ocurrido una falla durante la carga o la descarga de contenidos: el método **ioError**. Su uso es muy útil, ya que nos permite comunicarle al usuario si no se ha completado una carga exitosamente, a fin de que la reanude.

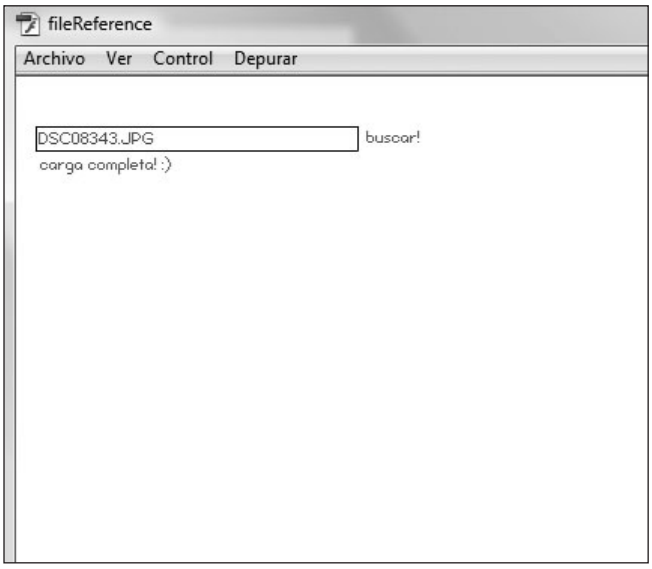


Figura 8. *Empleando el evento complete, sabemos cuándo ha concluido la carga del archivo.*

Si bien consideramos que hay contenidos que podemos dejar de lado y no abarcar en este libro, no era ese el caso de las clases **FileReference** y **FileReferenceList** y por ese motivo se encuentran en este apéndice. Este libro se basa en cómo utilizar la tecnología que pone a nuestro alcance Flash en casos prácticos aplicables a la industria multimedial de hoy en día, y en pleno auge de la Web 2.0, es en verdad necesario conocer de qué modo podemos utilizar Flash como cliente y cualquier lenguaje del lado del servidor para desarrollar aplicaciones que nos permitan generar la carga y descarga de contenidos, siendo éste uno de los grandes pilares de la Web como la estamos viviendo en la actualidad.

Servicios al lector

En esta última sección confeccionamos un índice que nos permitirá encontrar de forma más sencilla los temas que necesitemos. Además, ponemos a disposición un listado de sitios recomendados que nos ayudarán a mantenernos actualizados y a aumentar nuestros conocimientos.

SERVICIO DE ATENCIÓN AL LECTOR: usershop@redusers.com

ÍNDICE TEMÁTICO

SITIOS WEB

Porque no todo está en los libros, los sitios web que listamos a continuación son interesantes fuentes de información. Desde ya, la participación es bienvenida en los dos primeros sitios del listado, donde queremos hacer evidente la necesidad de conformar una comunidad de desarrolladores de habla hispana.

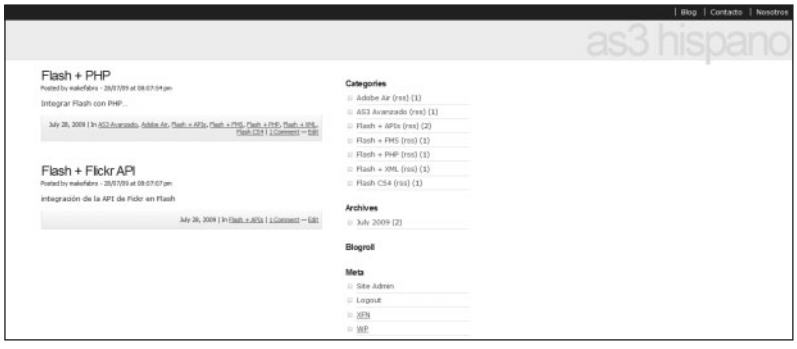
Actionscript hispano

www.facebook.com/home.php#/group.php?gid=92059349145&ref=ts

Este es un grupo de Facebook que tiene la intención de generar una comunidad de usuarios de habla hispana de Flash, ActionScript 3.0 y las plataformas relacionadas como Adobe AIR, Flex, Flash Media Server, etcétera; a fin de compartir conocimientos, experiencias, ideas, proyectos o, simplemente, el amor por esta plataforma.

AS3 Hispano

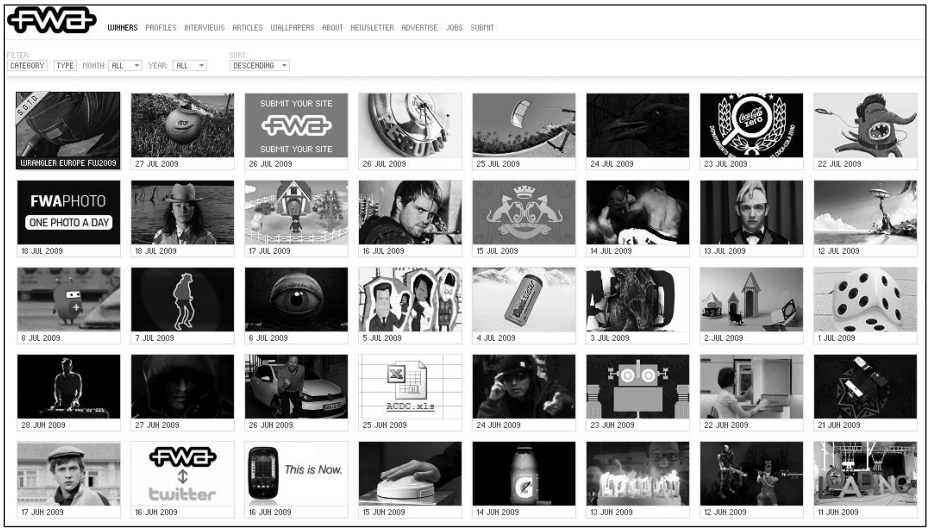
http://as3hispano.com/



Sitio web de Fabricio Mouzo y Mariano Makedonsky en el que los desarrolladores de este libro exponen tutoriales, notas y desarrollos sobre todo lo relacionado con el mundo Flash y ActionScript 3.0, el manejo de APIs, la integración de Flash con PHP y más.

The FWA

www.thefwa.com



The FWA (*Favourite Website Awards*) es un sitio que premia a diario lo que para ellos es el mejor sitio web del día. A su vez, también galardonan a la mejor web del mes y del año. Si bien no siempre vamos a estar de acuerdo con sus decisiones, es un buen lugar para conocer las últimas tendencias en desarrollo.

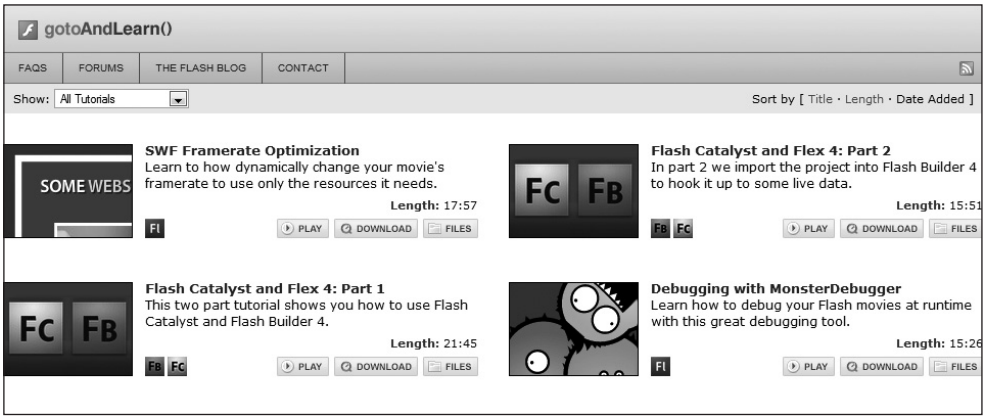
BlueVertigo

www.bluevertigo.com.ar

En este sitio podemos encontrar una enorme cantidad de recursos de utilidad para nuestros desarrollos y diseños; como sonidos, fuentes, iconos, stocks de fotografías, dibujos vectoriales y logos, entre otros muchos elementos.

GotoAndLearn()

www.gotoandlearn.com



Sitio web de Lee Brimelow, un excelente desarrollador que posee una gran virtud: transmite sus conocimientos con muchísima claridad. Nos ofrece videos tutoriales de temas sumamente interesantes que, si bien se encuentran en inglés, son muy fáciles de seguir y comprender. ¡Muy recomendable!

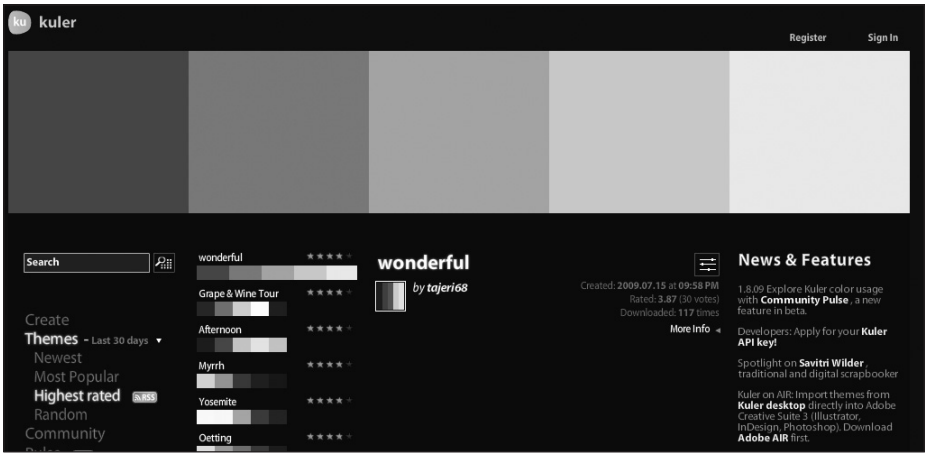
Andre Michelle

www.andre-michelle.com

Excelente desarrollador, principalmente en lo que hace al uso de sonido en Flash, a su representación visual y a su creación por medio de código. Vale la pena ver sus trabajos. Encontraremos todos sus proyectos en **http://lab.andre-michelle.com**. Entre ellos, el más interesante es www.hobnox.com/audiotool.1046.en.html.

Adobe Kuler

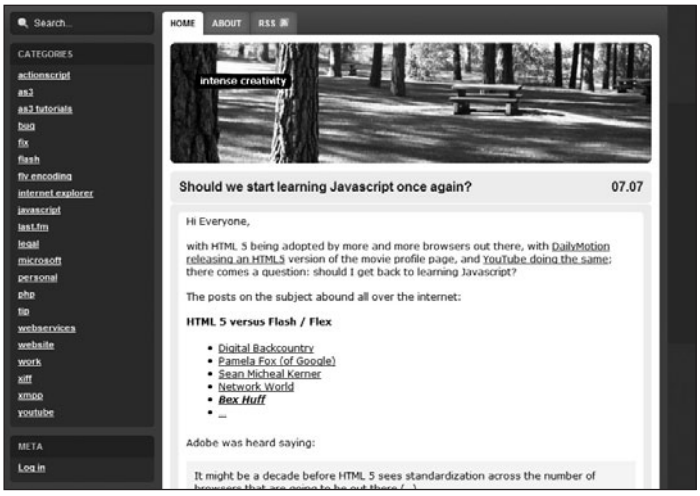
http://kuler.adobe.com



Un sitio web que es interesante por dos motivos. Por un lado, es un fiel reflejo de lo lejos que se puede llegar utilizando ActionScript 3.0 y, por otro, es una excelente herramienta de gran utilidad para generar paletas de colores. También cuenta con una aplicación desarrollada con Adobe AIR que recomendamos descargar.

Martin Legris Website

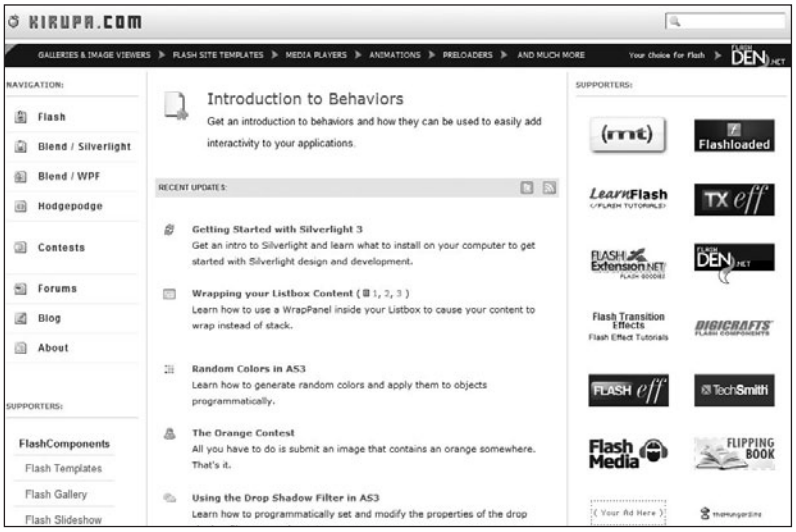
http://blog.martinlegris.com



En este blog encontraremos una cantidad interesante de recursos, principalmente en lo que hace a la implementación de video en Flash y a la integración de Flash con APIs de video de YouTube y de Vimeo. También hallaremos varios recursos de Flash, Flex, ActionScript y Adobe AIR, entre otras tecnologías.

Kirupa

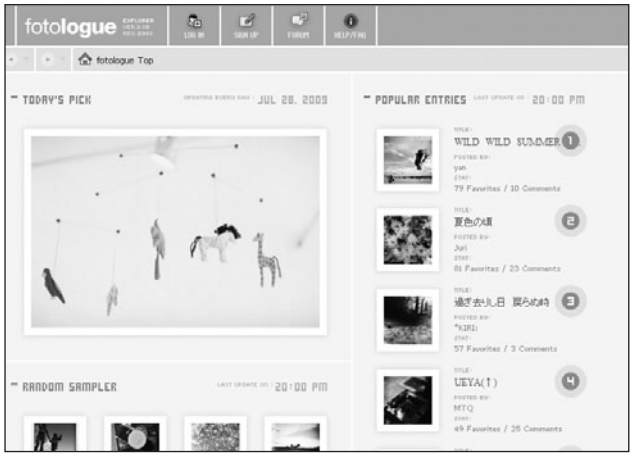
www.kirupa.com



Sitio web con una enorme cantidad de recursos para Flash, como templates, animaciones, preloaders, galerías de imágenes, reproductores, menús, interfaces, etcétera. Además hallaremos interesantes tutoriales sobre temas relevantes vinculados a Flash.

Fotologue

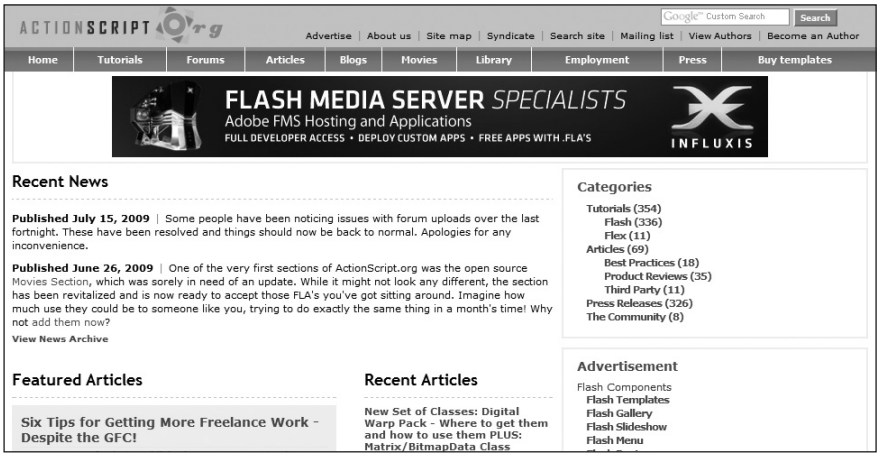
http://fotologue.jp



Un fotolog desarrollado íntegramente en Flash. No sólo es interesante por la calidad de sus imágenes y su limpio diseño de interfaz, sino que es un muy buen ejemplo del alcance de Flash, donde es notorio que, bien utilizado, puede implementarse para desarrollos de gran complejidad.

ActionScript.org

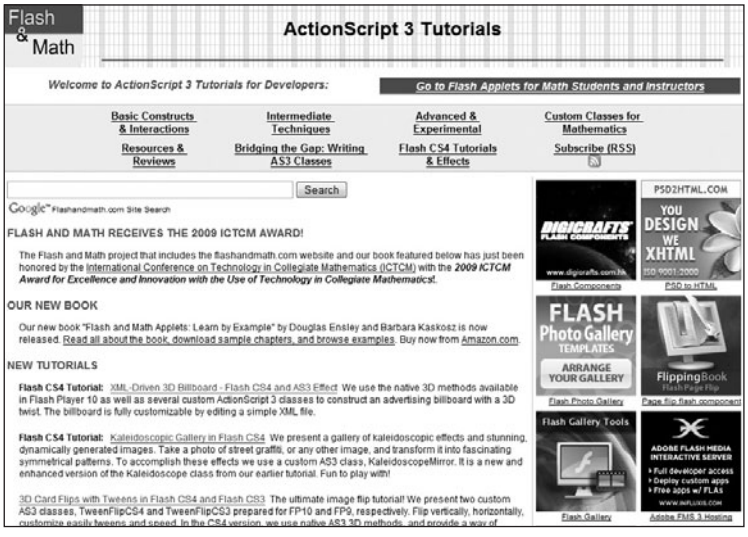
www.actionscript.org



Sitio web con una interesante cantidad de recursos, donde encontraremos tutoriales, foros, artículos, blogs y notas, entre muchas cosas más. Los temas están claramente diferenciados en categorías, siendo Flash y Flex los recursos más utilizados.

Flash & Math

www.flashandmath.com



En este sitio tenemos disponibles tutoriales de ActionScript 3.0. Lo interesante es que estos se encuentran distribuidos en tres categorías: básicos, intermedios y avanzados. También hay una cantidad importante de recursos y clases para reutilizar y desarrollos experimentales. Sitio muy completo, interesante y recomendable.

Byte Array

www.bytearray.org

Este es un sitio web para exigentes, de carácter experimental. Cuenta con una serie de desarrollos poco convencionales y hace uso de clases que sólo unos pocos se animan a utilizar en Flash. También brinda tutoriales y links. Sin dudas, la parte más interesante del sitio son sus proyectos.

Photoshop: Online Photo Editing

www.photoshop.com

Si creemos que Flash es útil únicamente para desarrollar pequeñas aplicaciones, reproductores o galerías de imágenes, es hora de ver este sitio. Photoshop.com es una web desarrollada íntegramente con Flash, Flex y ActionScript 3.0 que nos permite subir y editar fotos e importar imágenes empleando las APIs de otros sitios, entre muchas cosas más. Realmente imperdible.

GotoAndLearn() forum

www.gotoandlearnforum.com

Foro del sitio gotoAndLearn() donde, seguramente, encontraremos respuesta a muchas de las preguntas respecto a Flash, Flex y Adobe AIR. También hallaremos noticias, sitios de interés y la participación de mucha gente unida por el desarrollo.

Adobe Flash Player

www.adobe.com/es/products/flashplayer

Sitio web de Adobe Flash Player donde hallaremos información de utilidad, desde las últimas novedades y noticias hasta archivos PDF con detalles de los índices de penetración del plugin, hojas de datos, documentación, descarga de distintas versiones de Flash Player, entre otros temas de interés.

Claves para comprar un libro de computación.

1 Sobre el autor y la editorial

Revise que haya un cuadro "sobre el autor", en el que se informe sobre su experiencia en el tema.
En cuanto a la editorial, es conveniente que sea especializada en computación.

2 Preste atención al diseño

Compruebe que el libro tenga guías visuales, explicaciones paso a paso, recuadros con información adicional y gran cantidad de pantallas. Su lectura será más ágil y atractiva que la de un libro de puro texto.

3 Compare precios

Suele haber grandes diferencias de precio entre libros del mismo tema; si no tiene el valor en la tapa, pregunte y compare.

4 ¿Tiene valores agregados?

Desde un sitio exclusivo en la Red hasta un CD-ROM, desde un Servicio de Atención al Lector hasta la posibilidad de leer el sumario en la Web para evaluar con tranquilidad la compra, o la presencia de adecuados índices temáticos, todo suma al valor de un buen libro.

5 Verifique el idioma

No sólo el del texto; también revise que las pantallas incluidas en el libro estén en el mismo idioma del programa que usted utiliza.

6 Revise la fecha de publicación

Está en letra pequeña en las primeras páginas; si es un libro traducido, la que vale es la fecha de la edición original.

usershop.redusers.com

Visite nuestro sitio web

Utilice nuestro sitio **usershop.redusers.com**:

- Vea información más detallada sobre cada libro de este catálogo.
- Obtenga un capítulo gratuito para evaluar la posible compra de un ejemplar.
- Conozca qué opinaron otros lectores.
- Compre los libros sin moverse de su casa y con importantes descuentos.
- Publique su comentario sobre el libro que leyó.
- Manténgase informado acerca de las últimas novedades y los próximos lanzamientos.

También puede conseguir nuestros libros en kioscos o puestos de periódicos, librerías, cadenas comerciales, supermercados y casas de computación.

Compra Directa! usershop.redusers.com

usershop@redusers.com | (011) 4110.8700

Adquiéralo con todos los medios de pago*

• Capítulo Gratis • Avant Première • Promoción • Ofertas

(*) Sólo válido para la República Argentina

SuperBlogger

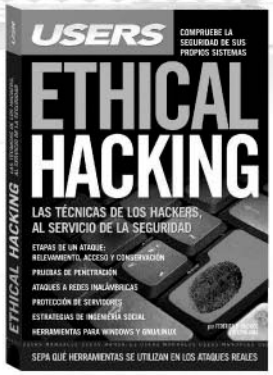
Esta obra es una guía para sumarse a la revolución de los contenidos digitales. En sus páginas, aprenderemos a crear un blog, y profundizamos en su diseño, administración, promoción y en las diversas maneras de obtener dinero gracias a Internet.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1347-96-4

UML

Este libro es la guía adecuada para iniciarse en el mundo del modelado. Conoceremos todos los constructores y elementos necesarios para comprender la construcción de modelos y razonarlos de manera que reflejen los comportamientos de los sistemas.

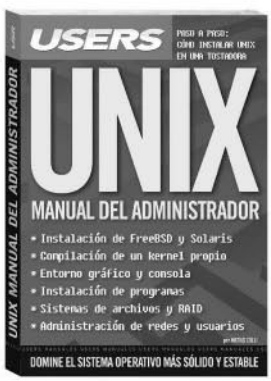
→ COLECCIÓN: DESARROLLADORES
→ 320 páginas / ISBN 978-987-1347-95-7



Ethical Hacking

Esta obra expone una visión global de las técnicas que los hackers maliciosos utilizan en la actualidad para conseguir sus objetivos. Es una guía fundamental para conseguir sistemas seguros y dominar las herramientas que permiten lograrlo.

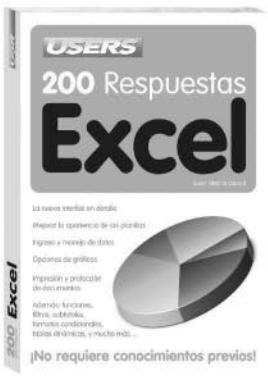
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-93-3



Unix

Esta obra contiene un material imperdible, que nos permitirá dominar el sistema operativo más sólido, estable, confiable y seguro de la actualidad. En sus páginas encontraremos las claves para convertirnos en expertos administradores de FreeBSD.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-94-0



200 Respuestas Excel

Una guía básica que responde, en forma visual y práctica, a todas las preguntas que necesitamos conocer para dominar la versión 2007 de Microsoft Excel. Definiciones, consejos, claves y secretos, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-1347-91-9

Hardware Extremo

En esta obra aprenderemos a llevar nuestra PC al límite, aplicar técnicas de modding, solucionar fallas y problemas avanzados, fabricar dispositivos inalámbricos caseros de alto alcance, y también a sacarle el máximo provecho a nuestra notebook.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-90-2

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.

Servicio Técnico de PC

Ésta es una obra que brinda las herramientas para convertirnos en expertos en el soporte y la reparación de los componentes internos de la PC. Está orientada a quienes quieran aprender o profundizar sus conocimientos en el área.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-89-6

Solución de Problemas PC

En esta obra encontraremos un material sin desperdicios que nos permitirá entender los síntomas que presentan los problemas graves, solucionarlos en caso de que algún imprevisto nos sorprenda y, finalmente, evitar que se repitan.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-1347-88-9

Diseño Gráfico

Esta obra es una herramienta imprescindible para dominar las principales herramientas del paquete más famoso de Adobe y para conocer los secretos utilizados por los expertos del diseño para trabajar de manera profesional.

→ COLECCIÓN: DISEÑO
→ 320 páginas / ISBN 978-987-1347-87-2

200 Respuestas Redes

Esta obra es una guía básica que responde, en forma visual y práctica, a todas las preguntas que necesitamos plantearnos para conocer y dominar el mundo de las redes hogareñas, tanto cableadas como Wi-Fi.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-1347-86-5

200 Respuestas Office

Una guía básica que responde, en forma visual y práctica, a todas las preguntas que necesitamos conocer para dominar la versión 2007 de la popular suite de Microsoft. Definiciones, consejos, claves y secretos, explicados de manera clara y didáctica.

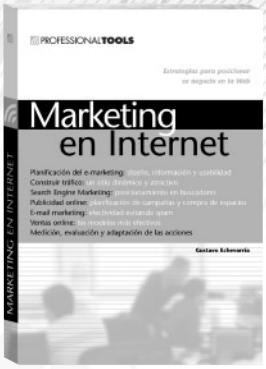
→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-1347-85-8

Finanzas con Microsoft Excel

Este libro es una obra con un claro enfoque en lo práctico, plasmado en ejemplos no sólo útiles sino también reales; orientada a los profesionales que tienen la necesidad de aportar a sus empresas soluciones confiables, a muy bajo costo.

→ COLECCIÓN: PROFESSIONAL TOOLS
→ 256 páginas / ISBN 978-987-1347-84-1

usershop.redusers.com



Marketing en Internet

Este libro brinda las herramientas de análisis y los conocimientos necesarios para lograr un sitio con presencia sólida y alta tasa de efectividad. Una obra imprescindible para entender la manera en que los negocios se llevan a cabo en la actualidad.

→ COLECCIÓN: PROFESSIONAL TOOLS
→ 288 páginas / ISBN 978-987-1347-82-7

200 Respuestas: Hardware

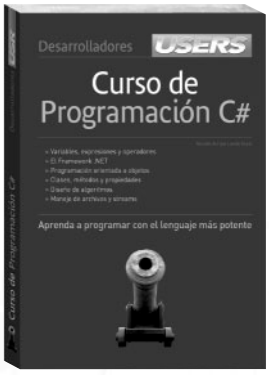
Esta obra es una guía básica que responde, en forma visual y práctica, a todas las preguntas que necesitamos hacernos para dominar el hardware de la PC. Definiciones, consejos, claves y secretos de los profesionales, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-1347-83-4

Curso de programación PHP

Este libro es un completo curso de programación con PHP desde cero. Ideal tanto para quienes desean migrar a este potente lenguaje, como para los que quieran aprender a programar, incluso, sin tener conocimientos previos.

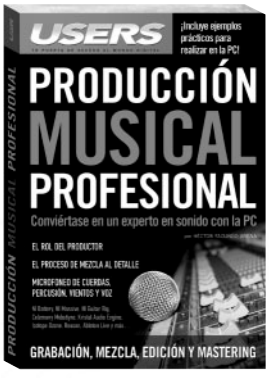
→ COLECCIÓN: DESARROLLADORES
→ 368 páginas / ISBN 978-987-1347-81-0



Curso de programación C#

Este libro es un completo curso de programación con C# desde cero. Ideal tanto para quienes desean migrar a este potente lenguaje, como para quienes quieran aprender a programar, incluso, sin tener conocimientos previos.

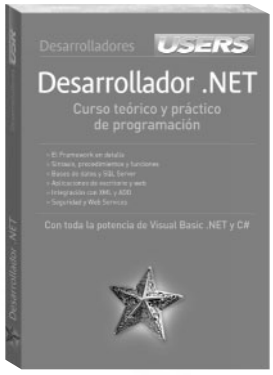
→ COLECCIÓN: DESARROLLADORES
→ 400 páginas / ISBN 978-987-1347-76-6



Producción musical profesional

Esta obra es un manual preciso y detallado que permite alcanzar la perfección a quienes quieren lograr el sonido ideal para sus composiciones. Está enfocado en el rol del productor, lugar desde donde construye los cimientos para producciones profesionales.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-75-9



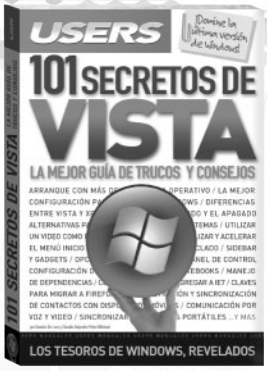
Desarrollador .NET

Ésta es una obra teórica y práctica para aprender a programar. Basado en el curso Desarrollador cinco estrellas de Microsoft, este material brinda las habilidades necesarias para iniciar el camino que nos lleve a convertirnos en desarrolladores de la plataforma .NET.

→ COLECCIÓN: DESARROLLADORES
→ 400 páginas / ISBN 978-987-1347-74-2

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.



101 Secretos de Vista

Una obra absolutamente increíble, con los mejores 101 secretos para dominar la última versión de Windows. En sus páginas encontraremos un material sin desperdicios, ideal para quienes quieren llevar el rendimiento de su PC al límite.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1347-80-3

Excel 2007: Guía de funciones

Este libro es una guía de referencia y consulta permanente que brinda acceso instantáneo a las funciones de Excel 2007, y sin duda se convertirá en un material indispensable para quienes quieran potenciar sus planillas de cálculo.

→ COLECCIÓN: MANUALES USERS
→ 368 páginas / ISBN 978-987-1347-79-7

El gran libro de la vida digital

Ésta es una obra visual y práctica, en la que aprenderemos a usar los principales dispositivos tecnológicos de la actualidad. Un libro fundamental para estar preparados y ser partícipes del cambio tecnológico que estamos viviendo.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-78-0

Electrónica digital

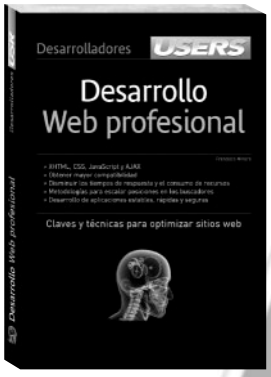
Una obra ideal para quienes desean conocer el mundo de la electrónica digital, y la simulación de circuitos y componentes. Un repaso de los principios de electricidad y electrónica, explicando en detalle las herramientas e instrumentos más importantes.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1347-73-5

200 Respuestas Fotografía digital

Esta obra es una guía básica que responde en forma visual y práctica a todas las preguntas que se nos plantean sobre la fotografía digital. Definiciones, consejos, claves y secretos de los profesionales, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-1347-71-1



Desarrollo Web profesional

Este libro resulta una herramienta imprescindible, de consulta permanente, que brinda las técnicas, claves y consejos que permitirán desarrollar sitios profesionales y perfeccionar nuestra labor en las principales tecnologías: XHTML, CSS, JavaScript y AJAX.

→ COLECCIÓN: DESARROLLADORES
→ 400 páginas / ISBN 978-987-1347-70-4

Armado Profesional de PC

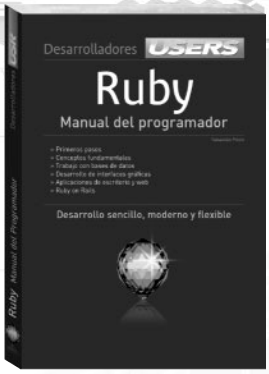
En las páginas de este libro, encontraremos cada procedimiento para aprender a armar computadoras de manera profesional. Todo detallado paso a paso, acompañado de consejos, técnicas y secretos de los máximos expertos en el área.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-69-8

Office 2007

Este manual ofrece un recorrido visual y práctico por cada aplicación de Office 2007, mediante explicaciones teóricas, guías visuales y procedimientos paso a paso. Con él aprenderemos a obtener el máximo provecho de todos sus programas.

→ COLECCIÓN: MANUALES USERS
→ 400 páginas / ISBN 978-987-1347-68-1



Ruby

Este manual presenta Ruby, uno de los lenguajes de programación más flexibles y poderosos de la actualidad para el desarrollo de aplicaciones de escritorio o web. Por su sencillez, es ideal para aprenderlo rápidamente, aun sin tener grandes conocimientos de programación.

→ COLECCIÓN: DESARROLLADORES
→ 432 páginas / ISBN 978-987-1347-67-4

PHP Master

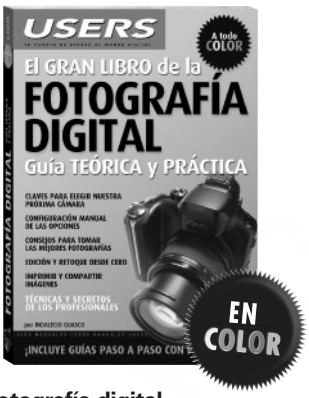
Este libro acerca al trabajo diario del desarrollador los avances más importantes incorporados en las últimas versiones de PHP. En sus páginas se exponen las técnicas actuales que permiten potenciar el desarrollo de sitios Web.

→ COLECCIÓN: DESARROLLADORES
→ 400 páginas / ISBN 978-987-1347-61-2

Reparación de componentes

En las páginas de este libro se detalla en profundidad el desarme de cada pieza de hardware, la metodología para realizar el diagnóstico y efectuar la reparación. Un material imperdible que incluye técnicas para convertirse en un profesional.

→ COLECCIÓN: MANUALES USERS
→ 240 páginas / ISBN 978-987-1347-58-2



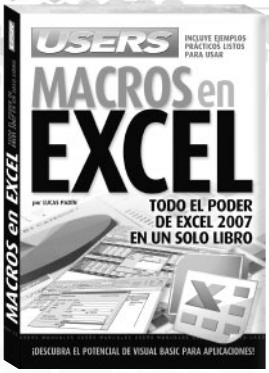
Fotografía digital

El objetivo de este libro es brindar las herramientas y los conocimientos necesarios para comprender a fondo la fotografía digital y aprender a realizar tomas con la calidad que nuestros recuerdos se merecen. ¡Con guías paso a paso y a todo color!

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-60-5

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.



Macros en Excel

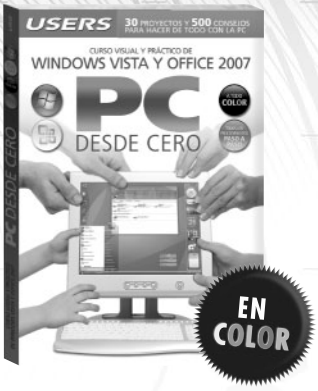
En las páginas de este libro encontraremos los conceptos teóricos y prácticos para construir macros e incluir código VBA en nuestras planillas. Una herramienta fundamental para quienes quieren conseguir un nivel profesional en sus trabajos.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1347-66-7

Reparación de impresoras

Este libro brinda las habilidades necesarias para convertirnos en técnicos de impresoras. En sus páginas encontraremos un recorrido visual al interior mismo de estos dispositivos, y aprenderemos todo lo que hay que saber acerca de las tecnologías disponibles en el mercado.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-64-3



PC desde cero

Domine el nuevo sistema operativo Windows Vista, aprenda todos los secretos de Excel, Word y PowerPoint, y convierta su computadora en un verdadero centro de entretenimiento multimedia. 500 actividades y consejos para hacer de todo con la PC.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-63-6

Reparación de PC

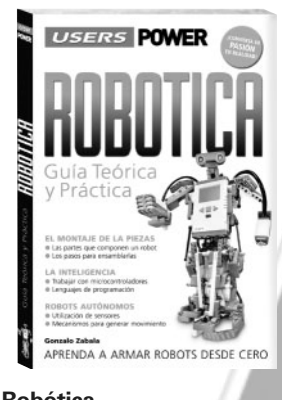
Las páginas de este libro ofrecen un recorrido visual y práctico por los principios de funcionamiento de cada dispositivo de la PC, las diferentes tecnologías, las claves para llevar adelante un diagnóstico certero y los pasos por seguir para efectuar la reparación.

→ COLECCIÓN: MANUALES USERS
→ 240 páginas / ISBN 978-987-1347-59-9

Photoshop para todos

Este manual desarrolla en forma teórica y práctica los conceptos necesarios para dominar Adobe Photoshop, la herramienta más poderosa para el retoque y la edición de fotografías. Además contiene cientos de consejos, para descubrir todo su potencial.

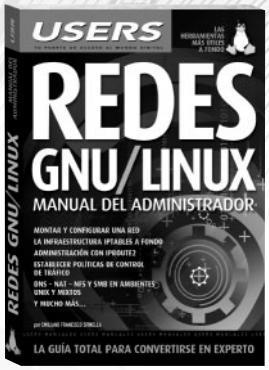
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-57-5



Robótica

En el interior de este libro encontraremos un recorrido teórico y práctico por cada uno de los conceptos fundamentales de la robótica, explicando en detalle cuáles son las partes que componen un robot, y los pasos por seguir para armarlo y programarlo a gusto.

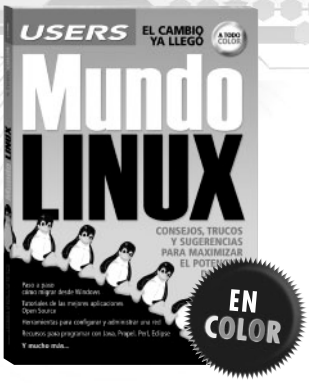
→ COLECCIÓN: MANUALES USERS
→ 288 páginas / ISBN 978-987-1347-56-8



Redes GNU/Linux

En las páginas de este libro encontraremos un recorrido por aquellos temas vinculados al armado de redes profesionales, y descubriremos cada una de las herramientas que GNU/Linux posee para su administración avanzada.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-1347-55-1



Mundo Linux

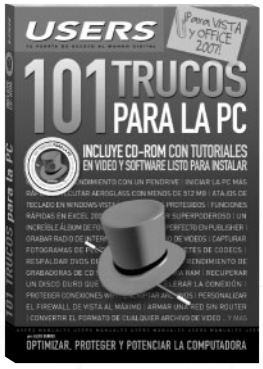
Este manual contiene informes, secretos, consejos y trucos que nos permitirán descubrir y aprovechar todo el potencial que posee este sistema operativo. Está destinado a usuarios que ya tienen alguna experiencia y desean dominar sus herramientas.

→ COLECCIÓN: MANUALES USERS
→ 256 páginas / ISBN 978-987-1347-54-4

Administrador de Redes

En este manual encontraremos en forma didáctica, clara y precisa todos los conceptos vinculados a la planificación, instalación y puesta en funcionamiento de una red en una empresa pequeña utilizando Windows Small Business Server.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-1347-53-7



101 Trucos para la PC

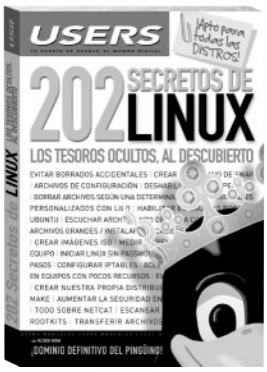
Este libro nos permitirá acelerar el funcionamiento de nuestra PC, convertirla en una fortaleza frente a los ataques, optimizar el rendimiento, navegar más rápido por Internet, aprovechar al máximo las nuevas características de Office 2007 y mucho más.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-1347-48-3

El gran libro del desarrollador

En este libro encontraremos el mejor material para estar actualizados con las últimas tecnologías, herramientas y recursos del mundo de la programación. 256 páginas a todo color donde los expertos desarrollan los conceptos fundamentales de cada plataforma.

→ COLECCIÓN: MANUALES USERS
→ 256 páginas / ISBN 978-987-1347-47-6

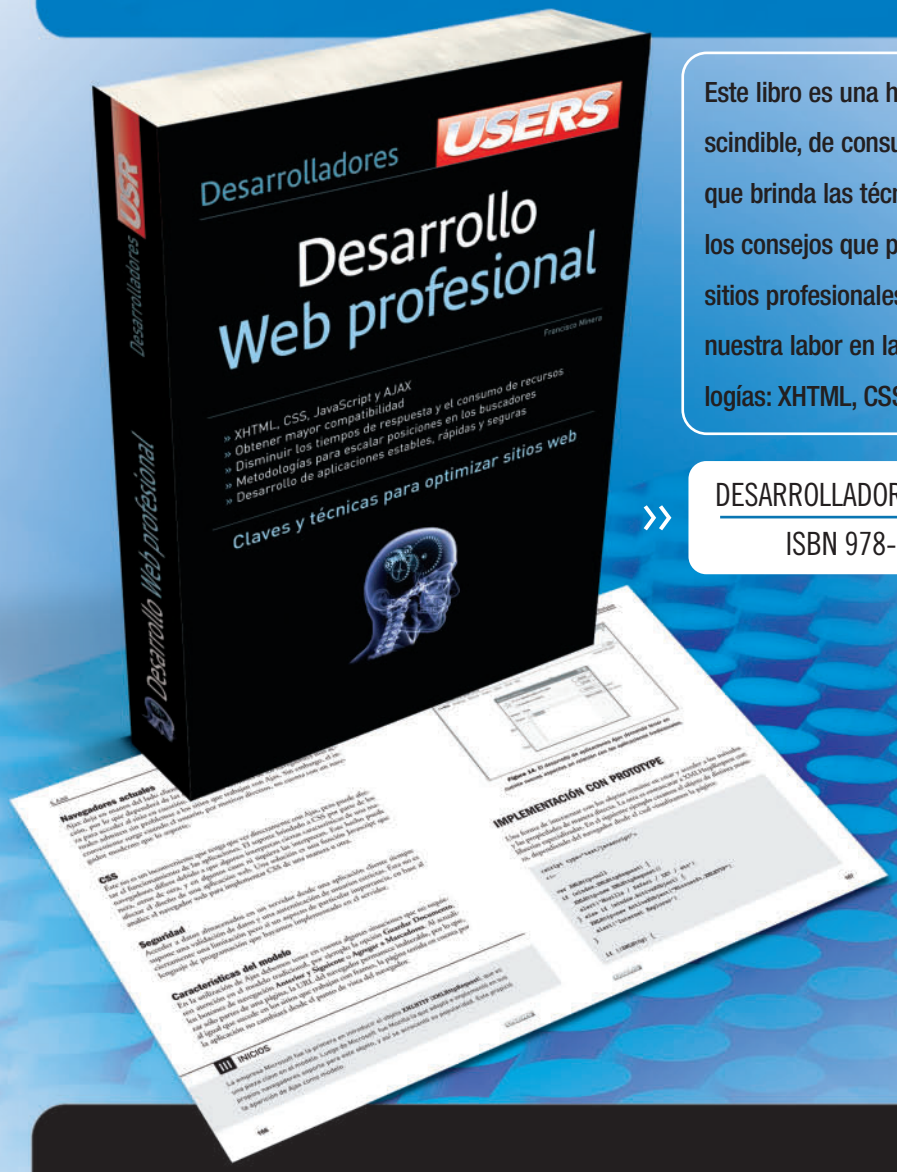


202 Secretos de Linux

Cómo optimizar el sistema y volverlo más seguro, aumentar el rendimiento del equipo, habilitar funciones especiales y automatizar tareas cotidianas, son sólo algunos de los objetivos que lograremos con los más de 200 secretos revelados en este libro.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-41-4

APRENDA XHTML, CSS, JAVASCRIPT Y AJAX



Este libro es una herramienta imprescindible, de consulta permanente, que brinda las técnicas, las claves y los consejos que permitirán desarrollar sitios profesionales y perfeccionar nuestra labor en las principales tecnologías: XHTML, CSS, JavaScript y AJAX.

DESARROLLADORES | 400 páginas
ISBN 978-987-1347-70-4

usershop.redusers.com

Adquiéralo con todos los medios de pago

• Capítulo GRATIS • Promoción • Ofertas •

USERS

CONTENIDO

NIVEL DE USUARIO

PRINCIPIANTE | INTERMEDIO | AVANZADO | EXPERTO

FLASH EXTREMO

Flash y ActionScript 3.0 se han convertido en dos de los recursos más empleados a la hora de desarrollar aplicaciones Web y, más recientemente, aplicaciones de escritorio. En esta obra comprenderemos todo su potencial y los beneficios que la plataforma nos propone, tanto a través de su implementación, como de la fusión con PHP, APIs, Flash Media Server 3.0 y Adobe AIR.

Por medio de proyectos prácticos, aprenderemos a desenvolvernos en entornos reales: carga de imágenes, uso de APIs, visualizadores de espectros de sonidos, reproductores de MP3 y video, integración con webcams y micrófonos, grabación de video con FMS3 y aplicaciones de escritorio con Adobe AIR, entre otros.

Una obra imperdible, que pone a nuestra disposición los recursos más empleados en la industria multimedia y nos garantiza estar a la vanguardia del desarrollo en tiempos de la Web 2.0.

redusers.com

En este sitio encontrará una gran variedad de recursos y software relacionado, que le servirán como complemento al contenido del libro. Además, tendrá la posibilidad de estar en contacto con los editores, y de participar del foro de lectores, en donde podrá intercambiar opiniones y experiencias.

FLASH EXTREME

An incredible work that helps us improve our knowledge of Flash CS4 and ActionScript 3.0, two of the most used multimedia resources. By reading this work you will be on top of the present technologies concerning the Web 2.0.

ISBN 978-987-663-009-2

MANUALES USERS MANUALES USERS MANUALES USERS MANUALES USERS MA

DESARROLLO DE APLICACIONES DE ALTO IMPACTO