USERS

EL MANUAL NECESARIO PARA CRECER

# FLASH DESARROLLO PROFESIONAL

## ALCANCE UN NUEVO NIVEL DE CONOCIMIENTOS EN ACTIONSCRIPT 3.0 Y FLASH CS5

USABILIDAD Y ACCESIBILIDAD

PROGRAMACIÓN ORIENTADA A OBJETOS

DESARROLLO ORIENTADO A DISPOSITIVOS

PERFORMANCE: OPTIMIZACIÓN DE MEMORIA Y FPS

SEO, REUTILIZACIÓN DE CLASES, ESCALABILIDAD DE PROYECTOS, PATRONES DE DISEÑO, ¡Y MUCHO MÁS!

por Fabricio Juan Mouzo Lema y Mariano Makedonsky




TÉCNICAS AVANZADAS Y BUENAS PRÁCTICAS

RedUSERS.com

# FLASH: DESARROLLO PROFESIONAL

ALCANCE UN NUEVO NIVEL
DE CONOCIMIENTOS EN
ACTIONSCRIPT 3.0 Y FLASH CS5

por Fabricio Juan Mouzo Lema, Mariano Makedonsky

RedUSERS

TÍTULO: Flash: desarrollo profesional
AUTOR: Fabricio Mouzo, Mariano Makedonsky
COLECCIÓN: Manuales USERS
FORMATO: 17 x 24 cm
PÁGINAS: 320



ISBN 978-987-1857-00-5

Mouzo Lema, Fabricio Juan
Flash: desarrollo profesional / Fabricio Juan Mouzo Lema y Mariano Makedonsky. - 1a ed. -
Buenos Aires : Fox Andina; Dalaga, 2011.
v. 221, 320 p. ; 24x17 cm. - (Manual users)
**ISBN 978-987-1857-00-5**
1. Informática. I. Makedonsky, Mariano II. Título

CDD 005


# ANTES DE COMPRAR

**EN NUESTRO SITIO PUEDE OBTENER, DE FORMA GRATUITA, UN CAPÍTULO DE CADA UNO DE LOS LIBROS EN VERSIÓN PDF Y PREVIEW DIGITAL. ADEMÁS, PODRÁ ACCEDER AL SUMARIO COMPLETO, LIBRO DE UN VISTAZO, IMÁGENES AMPLIADAS DE TAPA Y CONTRATAPA Y MATERIAL ADICIONAL.**

RedUSERS
COMUNIDAD DE TECNOLOGIA

redusers.com

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.

**LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL****

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com // usershop@redusers.com

# Fabricio Mouzo, Mariano Makedonsky

Fabricio Mouzo y Mariano Makedonsky son licenciados en Tecnología Multimedial, graduados en la Universidad Maimónides.

Juntos llevan a cabo el proyecto #90ED, a través del cual brindan soluciones de comunicación pensadas para distintos medios con un objetivo de experimentar y crear nuevas formas de interacción.

**Sitio Web**
www.90ed.net

**E-mails**
mouzofabricio@gmail.com
mariano.makedonsky@gmail.com

## Dedicatoria

A los incondicionales: la familia y los amigos.

## Agradecimientos

A la familia, a los amigos por el apoyo de siempre. A toda la gente que confía sus desarrollos a #90ED. A los docentes y a toda la gente de multimedia de la Universidad Maimónides. A Pedro Paleo y a María Ledesma, por guiarnos durante el desarrollo de esta obra; y a Nicolás Kestelboim, Diego Spaciuk y Claudio Peña, por hacer más sencillo nuestro trabajo desde la editorial. Un especial agradecimiento a Daniel Granatta, por escribir el prólogo de esta obra: el hecho de que una de las personas que más nos influyeron en nuestro crecimiento profesional hoy forme parte de estas páginas nos llena de orgullo. A Chuck, por compartir su sabiduría; y a Pachi, por compartir sus silencios. A todos los lectores de esta obra, que dedicaron unos minutos a esta página.

# Prólogo

Amado y denostado en partes iguales, Flash debe de ser, en software, lo que son las grandes y controvertidas estrellas musicales como Michael Jackson o Lady Gaga.

Amado, por sus seguidores, porque siempre ofreció la más completa solución a la diversidad de navegadores y versiones existentes en el mercado, con la complejidad que eso acarreaba, para conseguir que un mismo contenido pudiera ser visto de la misma manera por cualquier usuario. Denostado, por sus detractores, porque Flash siempre tuvo algo que podía ser reprochado, aunque, en general, cada una de esas carencias fuera solucionada en la siguiente versión. Siempre intentando abarcar todo lo posible en cuanto a la generación de contenidos para ser vistos en su pantalla, la historia de Flash es la de una carrera contra sí mismo, contraponiendo sus muchas virtudes a sus pocos defectos.

En este libro el lector encontrará la información necesaria para colocarse un nivel de conocimiento por encima; aquel en el que verá a Flash como una herramienta para dar cuerpo a ideas creativas.

Steve Jobs, en su célebre discurso de bienvenida a una de las generaciones de nuevo ingreso en la Universidad de Stanford de hace unos años, cuenta tres historias. En una de ellas narra una serie de hechos de su juventud, que derivaron en la creación de alguno de los maravillosos productos que su compañía lanzó años después. El relato, sin embargo, cobra sentido al explicar su protagonista que es mirando hacia atrás como uno reconoce y comprende la importancia que, para el momento presente, tiene cada uno de esos hechos del pasado, una serie de puntos interconectados en la vida de cada uno.

Y eso es justamente lo que es este libro: uno de esos puntos que enlazan lo que usted, lector, es hoy, antes de leer este libro, con lo que será dentro de cuatro, cinco o diez años, habiendo hecho muchas cosas con lo que haya aprendido en este texto. El qué es un misterio que le corresponde desvelar a usted, pero no podrá negar lo apasionante del reto. Buena lectura, ¡y buen viaje!

Daniel Granatta
*Director General Creativo JWT México*
**@danigranatta**

# El libro de un vistazo

El libro que tienen en sus manos propone ir un paso más allá en lo que respecta al desarrollo en ActionScript 3.0. Cuando maduramos profesionalmente, hace falta tener conocimientos que acorten esa brecha generada por la falta de información y escasez de recursos. Este libro pretende ser ese salto de calidad, apostando a pensar de manera profesional los desarrollos en todos sus aspectos, y aplicando los conocimientos adquiridos en la creación de un espacio de trabajo optimizado para futuros desarrollos.

## *01 CONSIDERACIONES INICIALES

Este capítulo propone hacer una revisión de lo que fue, de lo que es y de lo que será Flash. También se analiza la implicancia de nuevas tecnologías, como es el caso de HTML5, y la relevancia que cobra el inminente arribo de los dispositivos a la industria con la posibilidad de visualizar contenido generado por Flash.

## *02 ESTRUCTURA Y CONTENIDO PRELIMINAR

Antes de comenzar con la creación de un espacio de trabajo, es necesario hacer un relevamiento de las tecnologías, add-ons, plugins y clases disponibles que nos serán de ayuda a lo largo del resto del desarrollo y formarán parte de nuestro framework.

## *03 REUTILIZACIÓN Y ESCALABILIDAD

En este capítulo abarcamos uno de los temas más importantes de este libro: sentaremos las bases respecto a la manera de organizar los proyectos para que sus paquetes y clases sean reutilizables, el espacio de trabajo sea escalable, y sea posible contar con un orden que nos permita manejarnos de modo prolijo y fluido.

## *04 SEO EN FLASH: POSICIONAMIENTO

Analizamos aquí algunos de los recursos con los que contamos para hacer que nuestras películas Flash sean visibles para los motores de búsqueda: algunas técnicas son propias de la plataforma, y otras son externas, como es el caso de la creación de contenido alternativo utilizando JavaScript (SWFObject), HTML y PHP.

## *05 OOP Y DESIGN PATTERNS: ESTRUCTURA INTERNA

Al pensar en la creación de un entorno escalable y reutilizable, no podemos dejar de lado el tratamiento de dos conceptos imprescindibles para afrontar nuestros desarrollos de una mejor manera: cubrimos aquí las ventajas de la Programación Orientada a Objetos y la importancia de los patrones de diseño, en especial, de MVC y Singleton.

## *06 USABILIDAD Y ACCESIBILIDAD

En este capítulo explicamos las ventajas de emplear criterios de usabilidad y accesibilidad en los desarrollos a fin de brindar una mejor experiencia de usuario. Realizamos un abordaje de varias técnicas para obtener mejores resultados: desde SWFAddress para la navegación, hasta los sharedobjects de Flash para recordar el estado de la aplicación.

## *07 DESARROLLO EN FLASH PARA DISPOSITIVOS

En este capítulo realizaremos un abordaje de conceptos referidos al diseño de interfaces para dispositivos y de las implicancias que tuvo la salida de Flash Player 10.1 y 10.2. Analizamos las ventajas de contar con tantas plataformas aptas para el desarrollo y, a su vez, las limitaciones que cada una de ellas propone desde su entorno, prestaciones y recursos.

## *08 OPTIMIZACIÓN, MEMORIA Y FPS

Independientemente del tipo de desarrollo que llevemos a cabo, los conceptos de este capítulo son útiles ante cualquier situación que nos proponga Flash: veremos de qué manera optimizar procesos y obtener un buen rendimiento y performance, centrándonos en el rendimiento de los FPS y el consumo de memoria de nuestro desarrollo.

## *09 COMUNICAR DESARROLLO CON EL FRAMEWORK

Este capítulo presenta un cierre práctico, a través del cual, por medio de un ejemplo real, vemos cómo implementar en un único desarrollo todo lo que hemos incorporado y aprendido a lo largo de este libro. También veremos cómo queda articulado nuestro espacio de trabajo a fin de poder reutilizar su contenido en los proyectos generados.

## INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual podremos encontrar una serie de recuadros que nos brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados. Para poder reconocerlos fácilmente, cada recuadro está identificado con diferentes iconos:


CURIOSIDADES E IDEAS


ATENCIÓN


DATOS ÚTILES Y NOVEDADES


SITIOS WEB

# Contenido



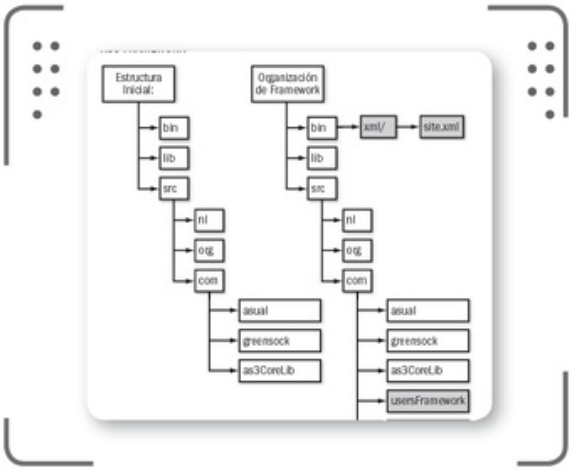
## *01 Consideraciones iniciales



## *02 Estructura y contenido preliminar



## *03 Reutilización y escalabilidad






## *04 SEO en Flash: posicionamiento



## *05 OOP y Design Patterns: estructura interna






## *06 Usabilidad y accesibilidad



## *07 Desarrollo en Flash para dispositivos

08

## Optimización, memoria y FPS






09

## Comunicar desarrollo con el framework



## Servicios al lector

# Introducción

En los últimos tiempos, se ha producido un importante cambio en la industria del desarrollo, y junto a estos cambios, se modificaron los usos de las tecnologías.

Las nuevas modalidades de desarrollo que surgen con HTML5, CS3 y JavaScript simplifican el entendimiento de lo que, para nosotros, representa la Plataforma Adobe Flash: cuando los recursos disponibles no satisfacen nuestras necesidades y los estándares no alcanzan para poder cumplir nuestros objetivos, asoma Flash como solución.

A Flash le corresponde brindarnos respuestas en áreas específicas, donde otras no pueden ofrecerlas: el desarrollo de sitios de alto impacto, el de videojuegos, la creación de RIAs y el uso de video en la Web. A nosotros nos corresponde elegir siempre la herramienta correcta para el desarrollo indicado, y no perder de vista que solo existen buenos o malos usos de las tecnologías, y delegar responsabilidades en una tecnología es eximirnos de las nuestras propias.

Cuando comenzamos a darle forma a este libro, pensamos en qué tipo de solución queríamos poner al alcance del lector: por lo general, los libros plantean soluciones a problemas específicos: cómo hacer galerías de imágenes, reproductores de audio o de video, etcétera. En la medida en que crecemos profesionalmente, precisamos de otro tipo de soluciones: aquellas que nos ayuden a pensar y a ver el desarrollo desde otra perspectiva, que nos permitan crecer y madurar como profesionales y nos den otro enfoque, pero por sobre todas las cosas, soluciones que nos ayuden a ahorrar tiempo de desarrollo, para que podamos invertirlo en aquello que realmente nos entretiene y nos llena: la creatividad.

Deseamos que las próximas páginas sean esa solución.

Mariano Makedonsky,
Fabricio Mouzo
**mariano.makedonsky@gmail.com**
**mouzofabricio@gmail.com**

# *01 –

# Consideraciones iniciales

Este primer capítulo tiene como objetivo acercar al lector cuáles son los usos que se planean para la **plataforma Adobe Flash**, cuál es el presente y el futuro de ActionScript 3.0, y su relación con otras tecnologías, principalmente, con HTML5. A su vez, analizaremos la relevancia que están cobrando los dispositivos móviles actuales.

# Destinatarios del libro

**Flash** y **ActionScript 3.0** ocupan una posición singular en el mercado; a diferencia de otros lenguajes de programación en los que la mayoría de los programadores están vinculados a las ciencias de la computación, muchos de los que manejan ActionScript aprendieron a utilizarlo como consecuencia inmediata de la necesidad de implementar líneas de código en los desarrollos llevados a cabo en la **IDE** (**Entorno de Desarrollo Integrado**) de Flash. Este vínculo directo que nace de la fusión de Flash y ActionScript hace que la aproximación de los desarrolladores a la plataforma en cuestión sea de lo más diversa: la gama de usuarios abarca un espectro tan amplio, que incluye a **diseñadores gráficos**, **artistas**, **animadores**, **programadores**, **diseñadores en comunicación visual** y **publicistas**, entre las más variadas ramas. Esta peculiaridad de la Plataforma Adobe Flash, si bien la consideramos sumamente enriquecedora por su carácter multidisciplinario, presenta una leve desventaja cuando los desarrollos se hacen cada vez más demandantes; en la mayoría de estas ramas, no se estudian fundamentos de programación, sino que los conocimientos que se van adquiriendo, en general, son de carácter empírico. Este libro tiene por finalidad acortar esa brecha, y acercar los fundamentos de la programación a aquellos desarrolladores que están sintiendo la necesidad de implementar soluciones profesionales con ActionScript 3.0.


**Figura 1.** `Flash`, `Flex` y `Air` conforman lo que de aquí en adelante denominaremos la **`Plataforma Adobe Flash`**.

# ¿Todos deben implementar el modelo de desarrollo que proponemos?

La Plataforma Adobe Flash propone un espectro inmenso de posibilidades para los programadores. En los últimos años, se diversificó en gran medida la cantidad de alternativas que surgen del uso de Adobe Flash y de ActionScript. Las opiniones tienden a ser de lo más variadas. Nuestro punto de vista al respecto es que **la manera correcta de llevar a cabo un desarrollo es aquella que propone ventajas sobre el resto**. Si sentimos que nuestros desarrollos están adquiriendo una mayor complejidad de forma gradual, este es el libro para afrontar ese paso de manera útil y práctica, y aprender a lidiar con proyectos de gran envergadura. Por el contrario, aquellos usuarios que utilizan Flash para animar o generar pequeñas aplicaciones deberían de seguir haciéndolo del modo en que lo han hecho hasta el día de hoy. La diversidad de Flash hace que todos puedan tener un lugar bajo la plataforma: desde los desarrolladores que llevan a cabo pequeños emprendimientos, hasta aquellos con emprendimientos de carácter profesional y de determinada complejidad. Los contenidos de este libro están dirigidos, claramente, al segundo de estos dos grupos, y da por sobreentendido que quien encara esta obra posee conocimientos sólidos sobre programación en ActionScript 3.0.

> LA PLATAFORMA ADOBE FLASH PROPONE INMENSAS POSIBILIDADES PARA LOS PROGRAMADORES

Indudablemente, con el correr del tiempo se va haciendo necesaria una aproximación más profesional a Adobe Flash, y se van a tener que tomar en consideración cuestiones que antes no eran tan determinantes, como la optimización de recursos y la performance.

## ADOBE LABS

En el sitio web que se encuentra en la dirección **http://labs.adobe.com** se publican las últimas novedades en las cuales están trabajando los ingenieros de Adobe. Es un buen lugar de referencia cuando queremos saber las innovaciones que aparecerán próximamente en la Plataforma Adobe Flash.


**Figura 2.** Las posibilidades que surgen en torno al desarrollo de contenidos considerando los productos de Adobe son amplias. Nosotros veremos el desarrollo profesional de aplicaciones.

# ¡Yo quiero programar! ¿Por qué leer todo esto?

En verdad, si lo que uno pretende es abocarse de entrada a la programación, se puede obviar este capítulo. Lo que buscamos plantear a través de esta pequeña introducción es el contexto bajo el cual nos encontramos actualmente y cómo este repercute de manera directa en nuestro modo de desarrollar. Hace algunos años, nuestro modo de pensar los desarrollos era completamente distinto del actual. Sin dudas, está cambiando la forma de utilizar las aplicaciones y los sitios que creamos. Antes de programar, debemos conocer la realidad. Una vez que sabemos de qué manera se están utilizando nuestras implementaciones, podemos abocarnos de lleno a la escritura del código, que, como veremos, va a diferir en gran medida de las

## IMPLEMENTACIÓN DE FLASH EN LA WEB

Debemos tener en cuenta que Flash no siempre fue implementado de la manera correcta Esto generó sitios y desarrollos con esta tecnología que tendrían que haber sido resueltos con otras. Nuestra manera de pensar en Flash es como aquella tecnología que debemos implementar cuando HTML no nos brinda los recursos necesarios para los requerimientos de nuestro desarrollo.

modalidades que se vinieron usando hasta ahora. La mayoría de los espacios de trabajo para Flash están pensados para implementar soluciones en la Web. Nuestro modo de pensar, es el de implementar un *framework* que nos brinde la flexibilidad necesaria para poder visualizar nuestros contenidos dentro de cualquier dispositivo. Para esto, algunas veces utilizaremos **ActionScript**; otras, **HTML5**; otras, **PHP**. Esto variará acorde a nuestras necesidades.

# Fundamentos de nuestro acercamiento a Flash

Vale la pena hacer algunas salvedades sobre nuestro punto de vista respecto al modo de pensar los desarrollos. Lejos de caer en los fanatismos, este libro intenta proponer soluciones. Creemos fervorosamente que Flash fue, es y seguirá siendo la solución a muchísimos problemas que se plantean hoy en día y forman parte de la industria del desarrollo. A continuación, enumeraremos algunos puntos para tener en cuenta, principalmente, a la hora de considerar la modalidad en la que planeamos pensar nuestros trabajos.

## ActionScript 3.0 y HTML5

**HTML5 y Flash no son rivales**; se trata de dos tecnologías que funcionan de manera complementaria. Flash, por su historia, es aquella plataforma que se implementa siempre que nuestras expectativas son mayores a lo que HTML propone como solución. Es decir, el uso que

### OPEN SCREEN PROJECT

En el sitio web que encontramos en la dirección **www.openscreenproject.org** se publica información respecto al emprendimiento **Open Screen Project** de **Adobe**, en el cual esta empresa trabaja en conjunto con las principales compañías del mundo (**Google**, **Sony**, **Intel** y **NVIDIA**, entre tantas más) para llevar a cabo la implementación de **FlashPlayer** en la mayor cantidad posible de dispositivos.

debería hacerse de Flash es aquel que sitúa las expectativas del desarrollador por encima de lo que HTML nos permite hacer. Una buena manera de pensarlo es la siguiente: si un desarrollo hecho en Flash puede ser reproducido en HTML sin mayores dificultades y logrando las mismas prestaciones, pues bien, en ese caso estamos haciendo las cosas mal, y el sitio debería haber sido desarrollado en HTML.


**Figura 3.** Este es, sin dudas, el error que suele cometerse: se considera a HTML5 y a Flash como competidores.

Como sabemos, HTML es el lenguaje de marcado de la Web. Por esta razón, como desarrolladores, tenemos la obligación y la necesidad de conocerlo, dominarlo y ser conscientes tanto de sus ventajas como también de sus limitaciones. Muchas veces, aun siendo desarrolladores de ActionScript 3.0, tendremos que utilizar HTML.

## ¿Qué cambia con HTML5?

Lo que propone **HTML5** es una serie de soluciones e implementaciones, entre las cuales podemos nombrar: audio, video, *drag and drop*, dibujo, uso de bases de datos locales y tags semánticos, entre otras. No disponíamos de estos recursos en las versiones anteriores de HTML, y en muchos casos, se tiende a creer que estas soluciones e implementaciones de HTML5 compiten con los recursos de Flash.

Esto, indudablemente, no es así. Desde luego, siempre son bienvenidas las mejoras que se realicen en cualquier lenguaje, porque esto va a generar un espacio para crear nuevas experiencias y

desarrollos. Pero estas mejoras con las que cuenta la versión 5 de HTML aún se encuentran muy por debajo de las posibilidades que brinda el uso de Flash. Un buen modo de pensar en las prestaciones de HTML5 es considerarlas como las que nos daba Flash cuando se encontraba tres o cuatro versiones atrás respecto a la actual.


**Figura 4.** En el sitio web **www.w3.org/html/logo** podemos descargar el logo de HTML5 en diversos formatos.

Si algún día HTML5 llegase a tener las prestaciones que podemos obtener hoy con **FlashPlayer 10** (lo que, de ocurrir, será dentro de varios años), Flash se encontraría en la versión 14 o 15, o la que correspondiera, haciendo aquello que hoy no se puede hacer. Porque justamente ese es el espíritu de Flash: elevar los **estándares**. Y eso mismo es lo que Flash hace: proponer alternativas en las cuales otros lenguajes presentan limitaciones o casi no pueden realizar dichas acciones. En definitiva, siempre es bienvenida la evolución de cualquier lenguaje, principalmente porque va a fomentar la aparición de nuevos recursos, incentivan la creatividad y proponen otra modalidad de hacer las cosas.

## CONTENIDO HTML5 EN NAVEGADORES

Existe un factor vital por el que debemos ser cautelosos respecto al uso de **HTML5**. Al momento de escribir estas líneas, el 60% de los usuarios aún no pueden visualizar contenidos desarrollados en HTML5, mientras que **FlashPlayer** presenta una penetración superior al 98%.

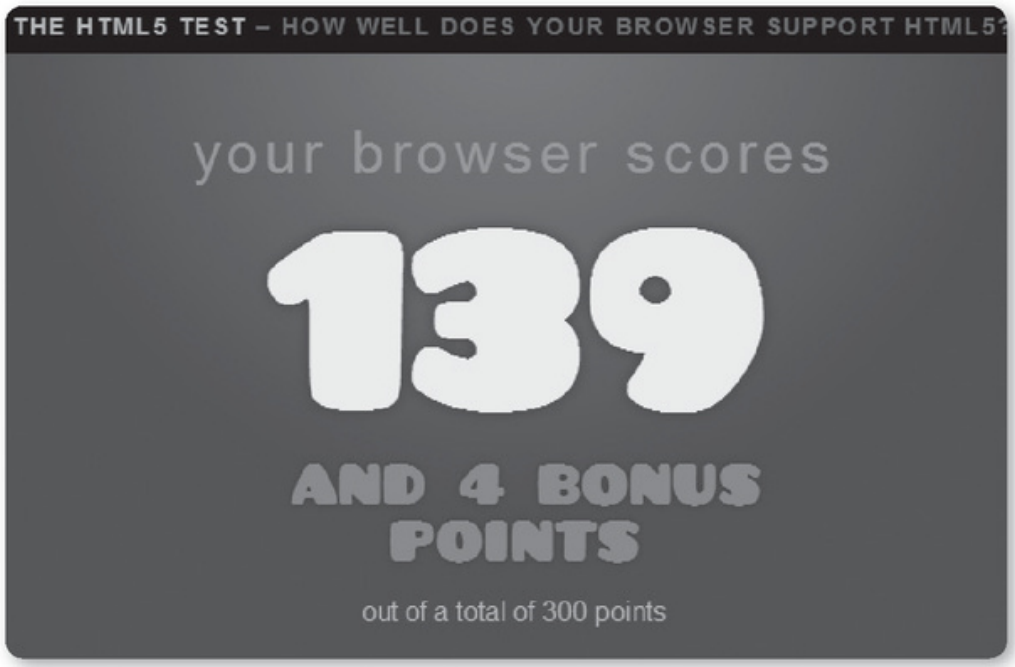



**Figura 5.** Al ingresar en el sitio que se encuentra en la dirección **`http://html5test.com`**, se puede obtener un informe de las prestaciones de **HTML5** con las que cuenta nuestro navegador.

## El futuro del video en la Web

Una de las principales causas por las que comenzó la controversia entre Flash y HTML5 fue la implementación del tag **<video>** en este último. De más está decirlo: Flash es mucho más que la mera reproducción de video. Lo que propone Flash es un entorno de reproducción avanzado, con una serie de prestaciones que se encuentran muy por encima de las posibilidades que brinda HTML5. ¿Esto quiere decir que el video en HTML5 es malo? No, al contrario, es

### VIDEO EN FLASH: EL ESPERADO STAGEVIDEO

En el sitio que se encuentra en **www.adobe.com/devnet/devices/articles/video_content_tv.html** se ofrece información sobre la clase **StageVideo**, por medio de la cual se hace uso de la **GPU** para procesar videos. Como resultado de esto, es posible obtener enormes beneficios en la performance, al renderizar videos de alta calidad con muy poco uso de la **CPU**.

una buena solución si lo que necesitamos es, simplemente, reproducir un video, sin mayores requisitos. Ahora bien, Flash ofrece una serie de ventajas, que son las que determinan cuándo debemos utilizar una tecnología u otra. Además de la posibilidad de reproducir video vía HTTP, Flash permite realizar lo siguiente:

- Reproducción de video en vivo.
- Reproducción de video bajo demanda.
- Protección de contenidos (DRM).
- Brinda la posibilidad de realizar la manipulación de los canales alfa del video.
- Personalización mucho más sencilla.
- Considerables mejoras en la performance desde la inclusión de StageVideo en FlashPlayer 10.2.

> FLASH NOS OFRECE VENTAJAS RELACIONADAS CON LA REPRODUCCIÓN DE VIDEO

## Códec h264, Plataforma Adobe Flash y lenguaje HTML5

Uno de los argumentos por los cuales muchos decretaron la superioridad de HTML5 sobre Flash respecto al uso de video en la Web fue, justamente, que HTML5 puede reproducir videos codificados en h264. Lo que la mayoría desconocía, es la capacidad de Flash de reproducir este códec, lo cual ocurre desde mediados del año 2007. Es imprescindible distinguir estos conceptos: **h264 es un códec**, **Flash es una plataforma** y **HTML es un lenguaje**. Es realmente positivo que HTML5 pueda reproducir videos y, tal vez, en algunos años sea la solución a todos los problemas en la Web, pero sus prestaciones hoy en día se encuentran muy por debajo de las de Flash en materia de video. A su vez, una gran ventaja que presenta el video en HTML5 es que

### YOUTUBE Y LA NECESIDAD DE FLASH

En el sitio web que encontramos en la dirección **http://apiblog.youtube.com/2010/06/flash-and-html5-tag.html** podemos consultar una nota escrita por el blog de **YouTube**, en la cual se brindan las explicaciones pertinentes con respecto a por qué tienen que implementar **FlashPlayer** en su sitio, y cuáles son las limitaciones que presenta **HTML5** para un sitio como YouTube.

podemos tener un plan alternativo de reproducción en aquellos casos en que un *smartphone* no permita reproducir contenido en Flash.


**Figura 6.** Debemos tener en cuenta que sitios de la talla de los famosos **YouTube**, **vimeo** y **hulu**, por sus características, necesitan las prestaciones de **FlashPlayer**.

# Smartphones

La cantidad de usuarios que visitan sitios desde dispositivos móviles se está multiplicando de manera exponencial. Si lo que nosotros buscamos es un espacio de trabajo que nos permita crear aplicaciones que se visualicen en todos los soportes, este es el punto por considerar. No podemos hacer caso omiso de HTML; por el contrario, debemos crear contenidos para todas las plataformas, y si bien Flash está creciendo gradualmente en lo que hace a su implementación en móviles, en muchos de estos equipos deberemos utilizar HTML.

En las próximas páginas veremos de qué manera detectar cuando un usuario está accediendo a un sitio desde un smartphone.

## PHONE MARKET SHARE

Si bien Adobe se encuentra trabajando en simultáneo junto a diversas compañías que se abocan al desarrollo de dispositivos móviles, actualmente solo corre Flash la versión **2.2** del sistema operativo **Android**, y el **market share** de este **S.O.** es solo del 17,2%. Este porcentaje es un buen indicio de que no podemos utilizar únicamente Flash en la creación de contenidos para móviles.

**Figura 7.** De la mano del lanzamiento de la versión 2.2 del sistema operativo de Google, **Android**, **FlashPlayer 10.1** hizo su aparición en los dispositivos móviles.

En este libro, veremos de qué manera crear desarrollos que se visualicen en la mayor cantidad posible de plataformas.

**Figura 8.** Tanto el **iPhone** como el **iPad** no cuentan con la posibilidad de reproducir contenidos creados con **FlashPlayer**.

## APLICACIONES PARA SMARTPHONES

En **http://help.adobe.com/es_ES/as3/mobile/index.html** se encuentra una guía oficial de Adobe, donde se analizan aquellas cuestiones que debemos tomar en consideración a la hora de desarrollar para dispositivos móviles: conservación de memoria y optimización de uso de CPU, entre otras cosas.

## Primeras conclusiones: teniendo en cuenta estas consideraciones, ¿cómo debemos desarrollar?

Este es un punto interesante para considerar. Tiempo atrás, pensábamos únicamente en la implementación de nuestro sitio teniendo en cuenta una computadora como dispositivo de visualización. En la actualidad, debemos pensar tanto en ella como en otros dispositivos móviles. Lo que debemos hacer es crear contenido independiente: uno para desktop y otro para móvil. Por esta razón, a lo largo de este libro, nos encargaremos de analizar la manera en que debemos diferenciar desde qué plataforma accede un usuario a nuestro sitio, una buena alternativa es mostrar el contenido adecuado para desktop dentro del dominio principal (**www.sitio.com**) y diseñar un sitio espacialmente para móviles y redireccionar a un subdominio (**www.mobile.misitio.com**).

> DEBEMOS CREAR CONTENIDO INDEPENDIENTE: UNO PARA DESKTOP Y OTRO PARA MÓVIL

## ¿La modalidad de desarrollo seguirá cambiando?

Indudablemente; y es por este mismo motivo que, tan importante como desarrollar, es tener la capacidad de adaptarse a los cambios. Por regla general, mientras contemos con más recursos, mayores serán nuestras posibilidades de encarar cualquier desarrollo y adaptarnos a los cambios, los cuales, muchísimas veces, resultan imprevisibles. El paradigma de desarrollar pensando en dispositivos móviles presenta una gran controversia.

Sin duda, nadie jamás imaginó que deberíamos evolucionar a un tipo de desarrollo donde hoy:

- Contamos con menos memoria.
- Contamos con menos procesador.
- Contamos con menos recursos.
- Contamos con menor resolución de pantalla.

Seguramente, dentro de unos años, lo que hemos planteado hasta aquí será solo un recuerdo, y los dispositivos móviles dispondrán de mayores recursos y prestaciones. Hoy, la realidad indica que si queremos abarcar el mayor espectro de posibles usuarios, debemos aprender a adaptarnos a estas limitaciones y desarrollar de acuerdo con las condiciones planteadas en la actualidad.

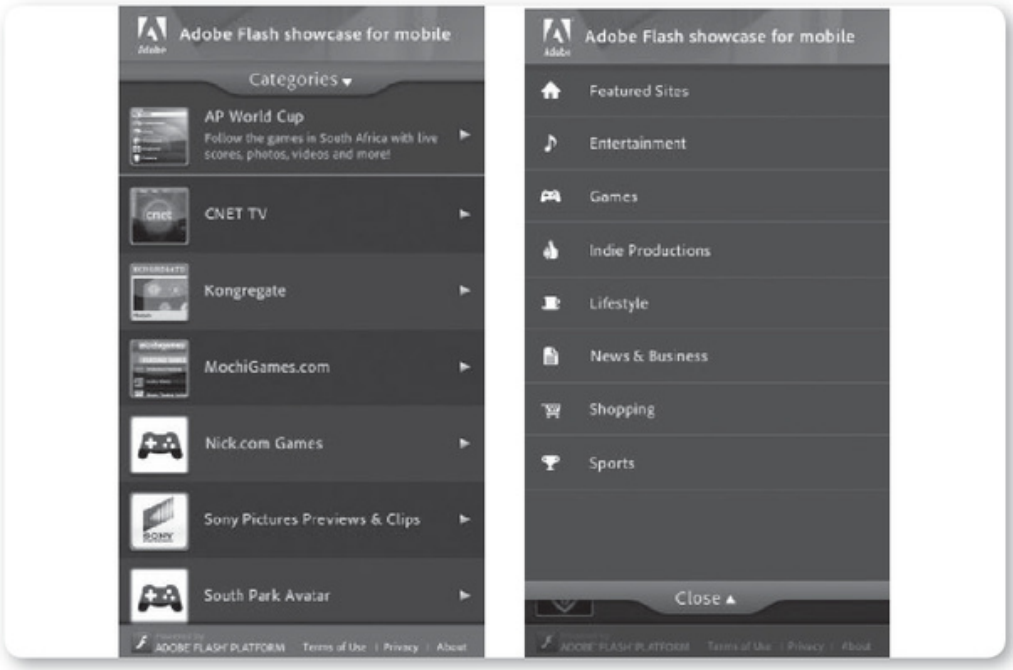



**Figura 9.** Si accedemos a **`http://m.flash.com`** desde nuestro smartphone, encontraremos un excelente ejemplo de un sitio desarrollado específicamente para ser visualizado en dispositivos móviles.

## RESUMEN

La intención de este primer capítulo es hacer un pequeño recorrido por la realidad del desarrollo actual y, de esta manera, tener una aproximación a los temas que se tratarán a lo largo de las próximas secciones. Las posibilidades de Flash y ActionScript son inmensas a la hora de pensar en un desarrollo escalable y, por su parte, la llegada de los *smartphones* nos obliga a pensar en nuevas alternativas y recursos al momento de implementar soluciones para la Web. De aquí en adelante, veremos cómo sacar ventaja de esta situación y de qué forma plantear un modo inteligente de desarrollo.

# Actividades

## TEST DE AUTOEVALUACIÓN

1. ¿Por qué debemos entender y manejar con claridad HTML?
2. ¿Por qué es importante que evolucione HTML5?
3. ¿Cuáles son las principales características de HTML5?
4. ¿Qué diferencia a Flash de HTML5?
5. ¿Cuáles son las ventajas que proporciona Flash sobre HTML5 en el uso de video?
6. ¿Por qué Flash y HTML5 son tecnologías complementarias, y no, competidoras?
7. ¿Por qué actualmente es importante pensar en los dispositivos móviles a la hora de llevar a cabo un desarrollo?
8. ¿Por qué es prematuro hacer desarrollos íntegramente en Flash, considerando la implementación en móviles?
9. ¿Por qué es prematuro hacer desarrollos íntegramente en HTML5, considerando las prestaciones de los principales navegadores?
10. ¿Por qué debemos tener en consideración todas estas cuestiones antes de comenzar a crear nuestro propio espacio de trabajo?

## ACTIVIDADES PRÁCTICAS

1. En vista de que aún no hemos realizado ningún ejercicio práctico y el propósito de este capítulo es efectuar un análisis de la situación actual del desarrollo, una idea interesante es buscar casos de buenas y de malas implementaciones. Un modo de averiguarlo es acceder a un mismo sitio desde una computadora y desde un dispositivo móvil, y ver si se hacen las implementaciones necesarias para que los sitios se visualicen correctamente en todos los dispositivos.

*02 –

# Estructura y contenido preliminar

Antes de crear nuestro espacio de trabajo, hay una serie de programas, add-ons, plugins y clases de ActionScript 3.0 que vamos a necesitar. Su implementación nos ayudará a mantener un orden, a optimizar nuestros desarrollos y a simplificar el trabajo. A su vez, organizar estos recursos nos permitirá sentar las bases de nuestros futuros desarrollos.

# Nuestro propio espacio de trabajo

Es importante saber que **espacio de trabajo** es sinónimo de **framework**. Vamos a hacer uso de estos términos en reiteradas ocasiones. Entendemos por espacio de trabajo o framework a un **conjunto de prácticas** y maneras de proceder que son comunes a la hora de resolver problemas de índole similar.

Pensémoslo de manera práctica: como desarrolladores, nos encontramos constantemente ante diversos proyectos por realizar. Si bien estos tienen particularidades (lo específico de cada desarrollo), hay algunas características que la mayoría de ellos tienen en común. Imaginemos el desarrollo de un sitio web. Debemos tener en cuenta que, en rasgos generales, ante un desarrollo de esta clase, deberíamos de considerar los aspectos que mencionamos a continuación:

- Criterios de usabilidad.
- Criterios de accesibilidad.
- Criterios para optimización de contenidos.
- Criterios para optimización de SEO.
- Tracking de contenidos.
- Debugging de contenidos.
- Testeo de las aplicaciones.

La gran ventaja de un espacio de trabajo radica en la posibilidad de agrupar aquellos elementos en común que vamos a utilizar en todos (o casi todos) ellos, para poder usarlos cuando la situación lo requiera, sin tener que programarlos otra vez cada vez que surja un nuevo desarrollo. Una vez que contemos con un entorno organizado, podremos usarlo como punto de partida para cualquier proyecto que hagamos y, así, optimizar **tiempos** y **recursos**. De esta manera, tendremos la posibilidad de abocarnos de lleno a la parte más importante de los proyectos, que son las especificidades de cada desarrollo y las características particulares que presenta cada uno de ellos.

Hay cuatro pilares fundamentales que sostienen un framework, y son los que evidencian la importancia de contar con uno cuando los desarrollos adquieren cierta complejidad. Un framework nos permite:

- Tener la posibilidad de que trabajen varias personas en un mismo proyecto de forma simultánea, sin que se superpongan en sus tareas o interfieran en las de los demás.
- Aislar la interfaz gráfica de la programación.
- Mantener una organización y una estructura que faciliten futuras modificaciones sobre los desarrollos.
- Contar con una estructura fácilmente escalable que nos permita hacer implementaciones, tanto en las generalidades del framework como en las particularidades de un desarrollo específico.

# ¿Cuál es el alcance del framework por desarrollar?

A la hora de crear un espacio de trabajo, sabemos que hay características comunes y particulares a cada tipo de desarrollo y que no disponemos de los mismos recursos al programar. Pensémoslo de manera práctica, para lo cual podemos imaginar que debemos encarar el desarrollo de un **videojuego**:

- Por un lado, las prestaciones serán distintas que las de un sitio web.
- Por otro, el alcance va a cambiar si se trata de un videojuego para la Web, de uno *stand alone* o de uno para smartphones.

La conclusión a la que arribamos con esto es que **el entorno y las prestaciones definen el alcance de un framework**. Por este motivo, consideramos que la mejor alternativa a la hora de encarar el desarrollo de un entorno de trabajo es limitarlo a un determinado entorno, y dentro de él, que este cuente con determinado alcance. Esto nos permitirá tener una mayor flexibilidad dentro y ser más específicos a la hora de hacer cualquier mejora o implementación. El espacio de trabajo

## ¿CREAR NUESTRO PROPIO FRAMEWORK?

Cada desarrollador tiene su modo de encarar un trabajo y de organizar sus contenidos. Hay frameworks gratuitos en Internet, que pueden ser de gran utilidad si los consideramos un punto de partida o los utilizamos para tomar ideas, pero generalmente terminamos por crear un entorno que se adapte a nuestras necesidades.

en el que vamos a manejarnos a lo largo de este libro está pensado para desarrollo web (ese sería nuestro entorno). Dentro de lo que abarca el desarrollo de contenido web, existen varios caminos. Nosotros nos centraremos en el de sitios (ese sería nuestro alcance).

Lo más importante de este libro no es enseñar la manera de hacer un espacio de trabajo de forma estricta, sino, por el contrario, presentar un modo **organizado**, **prolijo**, **útil**, **reutilizable** y **escalable** de hacer las cosas. Respetando esta modalidad de desarrollo, cuando surja la necesidad de encarar otro espacio de trabajo para otro tipo de desarrollo, esta obra podrá brindarles las herramientas para lograrlo.

En la siguiente sección nos encargaremos de realizar un listado de todo lo que vamos a necesitar antes de comenzar el desarrollo.

# Requisitos preliminares

Si bien dedicaremos el resto del libro a programar nuestras propias clases, debemos saber que existen varios recursos que precisaremos antes de adentrarnos en la escritura del código.

## Servidor local y servidor online

Para nuestro desarrollo, en algunas oportunidades debemos hacer uso de **PHP**. Del mismo modo, vamos a necesitar algún servidor online que nos permita testear los resultados en dispositivos móviles. Desde luego que PHP no es de importancia en este libro, pero es ventajoso conocer acerca de este potente lenguaje. Para llevar a cabo el desarrollo a nivel local, podemos emplear alguno de los siguientes **WAMPs** (**Windows**, **Apache**, **MySQL**, **PHP**) o **MAMPs** (**Mac**, **Apache**, **MySQL**, **PHP**).

### GAIA FRAMEWORK

**GAIA** es uno de los frameworks más utilizados. En la dirección web **www.gaiaflashframework.com** encontraremos información sobre él y sus características. Desde el mismo sitio es posible descargar y acceder a las novedades de la plataforma y a sus actualizaciones.

**WAMPS Y MAMPS DISPONIBLES**

| 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 algunos de los **WAMPs** y **MAMPs** disponibles.

# Desarrollo desde la IDE: Flash CS5

En aquellos casos en los que necesitemos hacer uso de la **IDE** de Flash, vamos a utilizar **Flash CS5** (aquellos lectores que cuenten con la versión **Flash CS4** también pueden utilizarla). No vamos a usar **Flex**. Quienes quieran llevar a cabo este desarrollo en él deberán hacer las adaptaciones necesarias en el código.

# Escritura de código: FlashDevelop

**FlashDevelop** es un editor de código gratuito y *open source*. Es la herramienta que vamos a utilizar para escribir los códigos necesarios en nuestro desarrollo. Si bien el editor de código de Flash CS5 se mejoró

**INSTALACIÓN Y USO DE WAMP Y MAMP**

Damos por sobreentendido que los lectores de este libro cuentan con conocimientos sobre PHP. En caso contrario, puede resultar conveniente leer algún texto que abarque estos temas con mayor profundidad; por ejemplo los libros *Proyectos Web: FLASH + PHP + XML + MySQL* y *Flash Extremo* de esta misma editorial pueden ser de gran utilidad en este caso.

considerablemente, FlashDevelop sigue siendo muy superior. Esta herramienta nos permite trabajar de un modo mucho más eficiente y administrar proyectos de una forma más ordenada en comparación con Flash , por esta razón es necesario considerarla para nuestros proyectos.

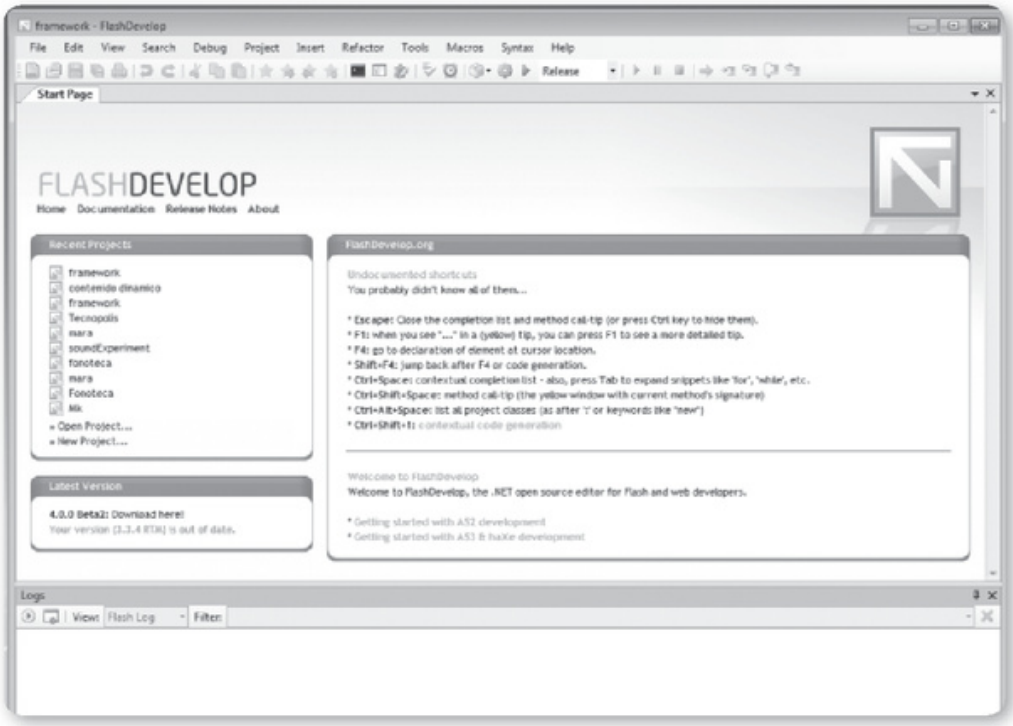



**Figura 1.** En esta imagen vemos la interfaz de **FlashDevelop**. Haremos uso de ella a lo largo de todo el libro.

## Usuarios de Mac

Lamentablemente, FlashDevelop no cuenta con una versión para Mac. Al ser una tecnología desarrollada con **Microsoft Windows .NET 2.0**,

### DESCARGA DE FLASH CS5

En caso de no contar con **Adobe Flash CS5**, es posible descargar una versión trial del producto desde la siguiente dirección web: **www.adobe.com/cfusion/tdrc/index.cfm?product=flash**. Tengamos en cuenta que esta versión es completamente funcional durante el tiempo de prueba.

funciona solo en sistemas operativos de la familia Windows (a partir de la versión **XP** de este sistema operativo). Aquellos lectores que sean usuarios de sistemas Macintosh deberán desarrollar desde la IDE de Flash, o bien buscar algún editor de código que pueda correr en sus sistemas operativos. El editor **FDT**, si bien es pago, puede ser una alternativa. En su sitio web hay una versión de prueba.

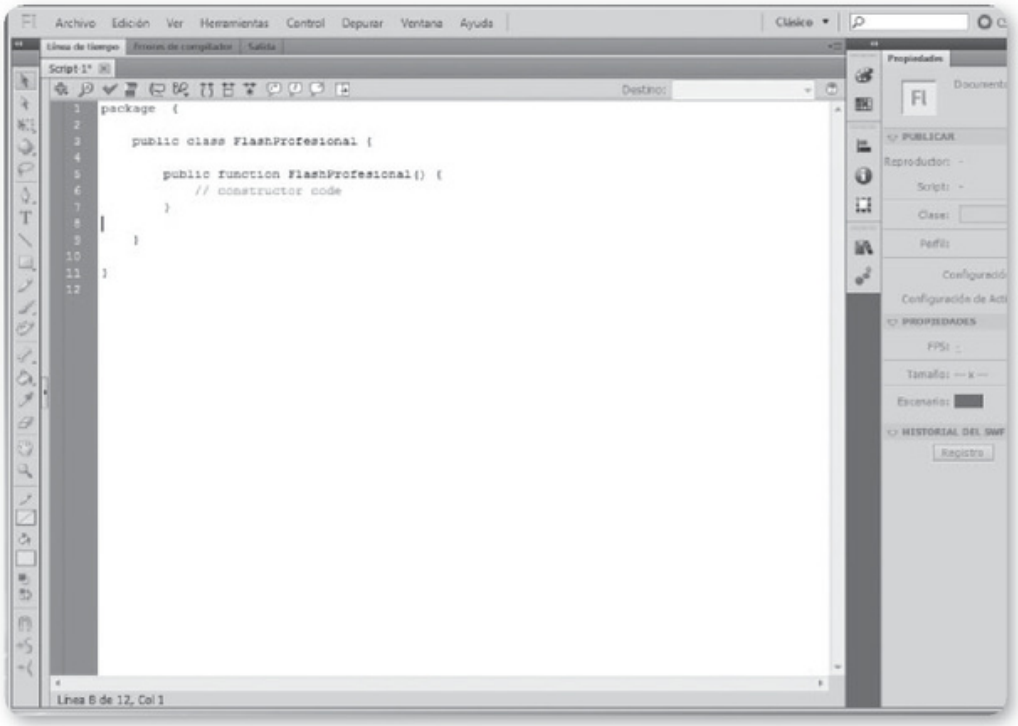



**Figura 2.** Buenas noticias para quienes utilizan el editor de código de Flash CS5: se introdujeron importantes mejoras en su interfaz.

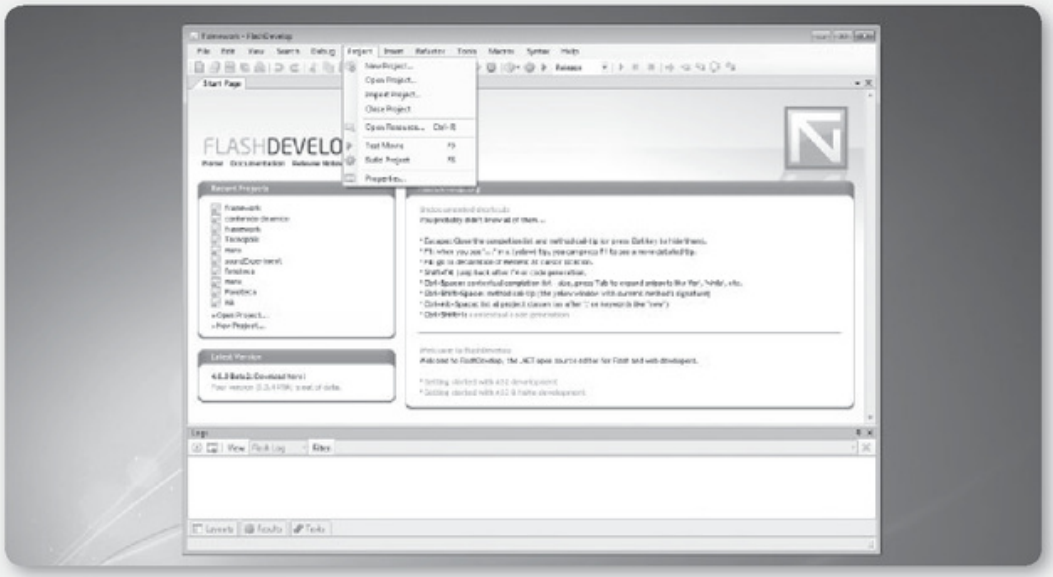
Una vez que descargamos FlashDevelop y lo instalamos en nuestro sistema, debemos crear un nuevo proyecto.

## DESCARGA DE FLASHDEVELOP

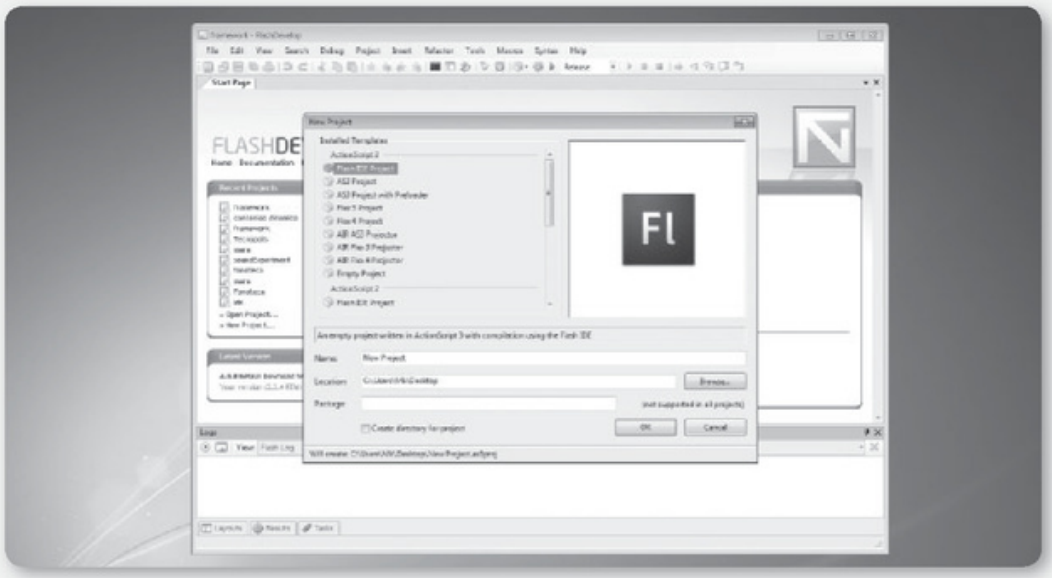
Podemos descargar cualquiera de las versiones de **FlashDevelop** desde el sitio web que encontramos en la siguiente dirección: **www.flashdevelop.org**. A su vez, allí es posible que encontremos información relevante sobre la plataforma y la documentación completa de ella.

## CREAR UN NUEVO PROYECTO CON FLASHDEVELOP

**01** En primer lugar, vaya a la pestaña **Project** y seleccione la opción **New Project...**.



**02** FlashDevelop abrirá una nueva ventana, indique el nombre del proyecto (**Name:**) y su ruta (**Location:**). En este caso, es recomendable crear el proyecto dentro del servidor local; esto le permitirá hacer uso de PHP cuando la situación lo requiera.



Luego de esto, tenemos nuestro primer proyecto. Dentro del **Project** deberíamos de ver la estructura que se presenta en la **Figura 3**.

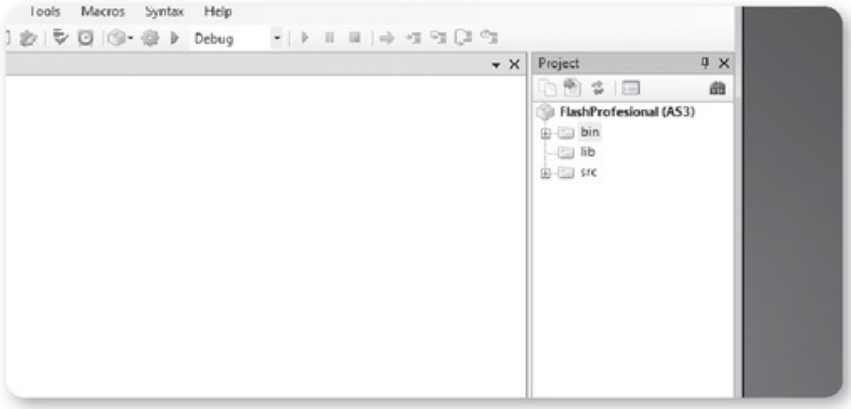



**Figura 3.** Una vez creado el proyecto, FlashDevelop genera automáticamente las carpetas **bin**, **lib** y **src**.

# Organización de proyectos

Al crear un proyecto, FlashDevelop crea automáticamente tres carpetas dentro del directorio que corresponde a nuestro proyecto: **src**, **lib** y **bin**.

## src

Dentro de **src** almacenaremos nuestros códigos fuente (**src** es la abreviatura de *source*). Aquí dentro organizaremos las clases que vamos a crear (**.AS**), y aquellos paquetes de clases que usaremos e iremos descargando en las próximas páginas.

### FLIXEL FRAMEWORK

Una de las características de los frameworks es que su organización y funcionamiento son acordes a los objetivos que se persiguen. Es por esto que no existe una única clase de espacio de trabajo. El framework **Flixel** (**http://flixel.org**) está claramente orientado al desarrollo de videojuegos por medio de ActionScript 3.0

### bin

Dentro de la carpeta **bin** vamos a guardar nuestras películas **.SWF**. Es sumamente importante mantener separados los códigos fuente de nuestro desarrollo (**src**), de la publicación (**bin**) correspondiente. A su vez, dentro de esta carpeta almacenaremos todos aquellos directorios, subdirectorios y archivos que forman parte del proyecto. En definitiva, si se trata de un proyecto web, esta carpeta contendrá todo lo que deberemos subir al servidor.

### lib

El directorio **lib** almacena los archivos **.FLA** de nuestro proyecto.

Concluido este paso, contaremos con la estructura básica del proyecto. Ahora bien, es necesario comenzar a volcar el desarrollo dentro de él. Como dijimos anteriormente, la carpeta **src** será el lugar en el cual almacenaremos los códigos fuente: paquetes que contienen clases que utilizaremos en el trabajo. Algunas de esas clases serán creadas a lo largo del libro y otras las vamos a descargar. Pero antes de comenzar a colocar clases aquí dentro, analizaremos otros aspectos imprescindibles del entorno de trabajo.

## Navegadores, plugins y add-ons

Es de esperar que todo desarrollador ya esté familiarizado con las herramientas que tenemos a nuestro alcance en cuanto a navegadores, *plugins* y *add-ons* a la hora de desarrollar. Simplemente, hacemos esta pequeña distinción porque vamos a necesitar algunos requisitos para llevar a cabo nuestro desarrollo sin complicaciones.

### Navegadores

Si bien la decisión de qué navegador utilizar corre por parte de cada lector, nosotros recomendamos implementar el uso de **Mozilla Firefox** a la hora de desarrollar, debido a la gran cantidad de extensiones que nos permite agregar; a su vez, conviene contar con el resto de los navegadores para testear los contenidos en cada uno de ellos (al menos habrá que hacerlo en los navegadores web más utilizados (**Internet Explorer**, **Firefox**, **Google Chrome**, **Safari** y **Opera**).

## Add-ons: firebug y flashTracer

Si bien son excluyentes y se pueden obviar, contar con ellos puede brindarnos ventajas a la hora de desarrollar. El primero es **Firebug** (**https://addons.mozilla.org/es-ES/firefox/addon/1843**), una gran herramienta que nos permite hacer *debug*, monitorear nuestras aplicaciones; y analizar estructuras CSS, HTML y JavaScript. El segundo es **FlashTracer** (**https://addons.mozilla.org/en-US/firefox/addon/3469**), que nos permitirá tener fácil acceso a los mensajes de salida de Flash desde el propio navegador web.

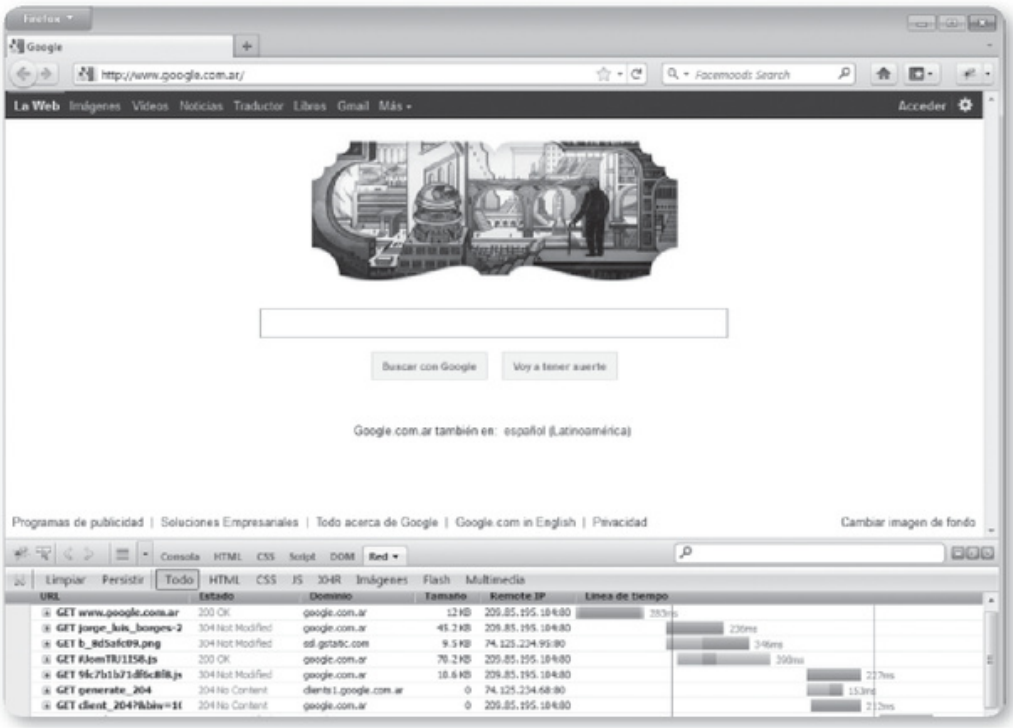



**Figura 4. `Firebug`** instalado dentro de nuestro navegador. Vemos los archivos que se están cargando dentro de él en la parte inferior.

### ORGANIZACIÓN DE CONTENIDOS

Si bien FlashDevelop propone una manera de organizar proyectos, existen varios modos de hacerlo. Lo que en verdad es importante no son los nombres de los directorios, sino el hecho de mantener un orden y tener por separado lo que son los códigos fuente del desarrollo, de la publicación.

## Plugins: Flash Player Debugger Version

Existe una serie de versiones de Flash Player denominadas Flash Player Content Debugger, que fueron creadas para las plataformas Windows, Mac y Linux. Contar con ellas nos permitirá hacer debug de los contenidos de nuestra película.

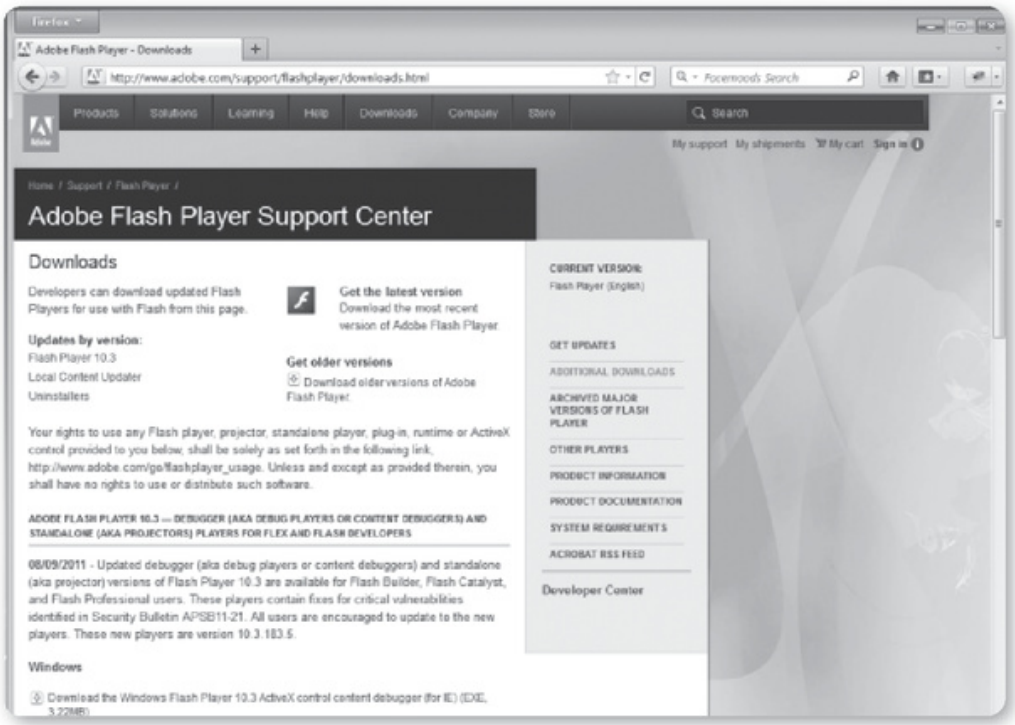



**Figura 5.** En **`www.adobe.com/support/flashplayer/downloads.html`** podemos encontrar **`Flash Player`**.

# Debugging de contenidos

Se denomina debugging al proceso mediante el cual depuramos nuestra aplicación, a fin de encontrar errores o defectos en ella; a su vez, es de gran ayuda cuando necesitamos saber qué mensajes se están imprimiendo a través del *output* de Flash (las acciones que surgen como resultado de la función **trace()**). Dentro del entorno de la Plataforma de Abobe Flash podemos leer los mensajes de salida, pero muchas veces tenemos que probar nuestras aplicaciones dentro del navegador. Flash no nos brinda las herramientas para leer estos mensajes de salida, pero, existen varias maneras de hacerlo. Veamos algunas alternativas:

## FlashTracer

Este add-on nos da la posibilidad de acceder desde el navegador a los mensajes de salida generados por medio de la función **trace()** de Flash. Una vez que instalamos la extensión dentro de nuestro navegador, FlashTracer es una buena alternativa para aquellos casos en los que no tenemos demasiadas pretensiones con respecto a la información que nos brindan los mensajes de salida. Puede sernos útil en proyectos pequeños, y su ventaja es que no debemos hacer uso de clases externas desde nuestro código para ver los resultados: simplemente, los vemos utilizando la función **trace()** dentro de Flash.

## MonsterDebugger

**MonsterDebugger** es un debugger desarrollado para Flash, Flex y Air, de carácter *open source* o libre y que presenta una serie de ventajas sobre la anterior alternativa, FlashTracer:

- Muestra mensajes de salida completamente detallados. Nos permite ver **cadenas de texto**, **números**, **estructuras XML**, **arrays** y **objetos**, con detalles de cada uno de ellos.
- Permite edición de contenido en vivo. Podemos modificar propiedades de nuestra aplicación desde el mismo debugger.
- Presenta la estructura de árbol de nuestra aplicación. Nos permite conocer la estructura de nuestro proyecto con gran nivel de detalle.

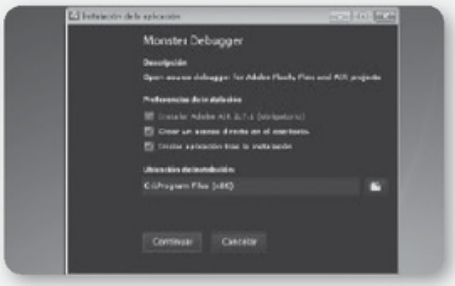
La descarga de la aplicación puede hacerse desde la sección **Downloads** de su sitio web: **http://demonsterdebugger.com**. Veremos de qué manera instalarlo y agregarlo a nuestro proyecto para hacer uso de él a lo largo de nuestro desarrollo.

### VIDEOTUTORIAL DE MONSTERDEBUGGER

En el sitio **gotoandlearn.com** hay un tutorial acerca de cómo instalar **MonsterDebugger**. Es posible acceder a él desde el link **www.gotoandlearn.com/play.php?id=109**. Dentro de este sitio se ofrece una gran cantidad de videotutoriales que son de enorme ayuda. El sitio y los videos se encuentran en inglés, pero pero es probablemente uno de los mejores recursos para comenzar a adentrarse en temas más complejos que hacen al mundo Flash, Flex, AIR, JavaScript, HTML5, entre otras tecnologías.

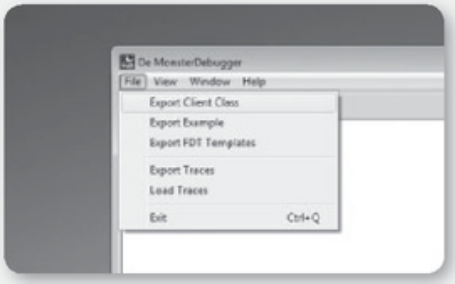
## DEBUGGING CON MONSTERDEBUGGER

**01**




Una vez descargado el programa, deberá proceder a su instalación. Defina las **preferencias de instalación** y la ubicación para la aplicación correspondiente (**ubicación de instalación**).

**02**




Cuando el proceso concluya, se abrirá el programa. Desde allí, puede descomprimir la clase de ActionScript 3.0 que le permitirá usar esta interfaz. Con este fin, haga clic en **File** y, luego, en **Export Client Class**.

**03**

Se abrirá un cuadro de diálogo dentro del cual debe definir la ruta en la que quiere descomprimir la clase cliente. Como se mencionó anteriormente, todas las clases por utilizar deben incluirse en la carpeta **src** del proyecto.

**04**

Una vez que haya concluido el paso anterior, se le informará que la clase se descomprimió correctamente. Verá que, dentro del directorio **src**, cuenta con un nuevo directorio llamado **nl**. En él están los paquetes y la clase necesarios para hacer uso de esta interfaz en el debugging.

A continuación veremos de qué manera hacer uso del output de Flash. Abrimos un nuevo archivo de ActionScript 3.0, lo guardamos en la carpeta **src** y agregamos el siguiente código:

```
import nl.demonsters.debugger.MonsterDebugger;

var myInstance:MonsterDebugger = new MonsterDebugger(this);

MonsterDebugger.trace(this, "nuestro primer trace con MonsterDebugger! :)");
```

En la primera línea, lo que hacemos es importar la clase **MonsterDebugger**. Luego, para usarla, debemos crear una instancia con el código que se muestra a continuación:

```
var myInstance:MonsterDebugger = new MonsterDebugger(this);
```

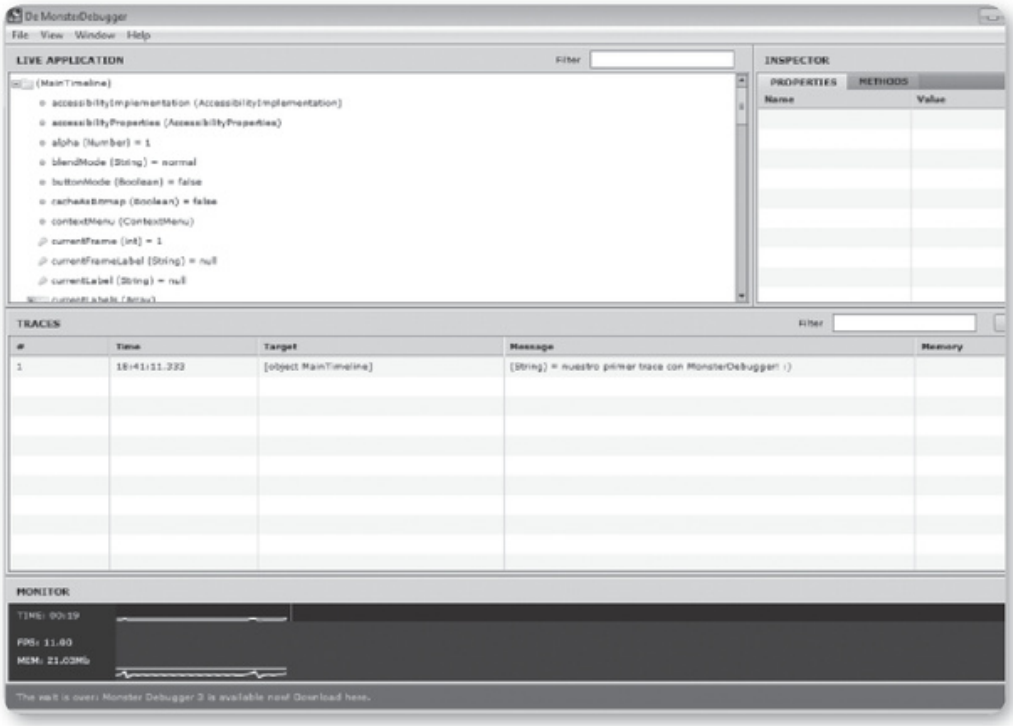



**Figura 6.** Si observamos en los **TRACES** de la interfaz gráfica de **MonsterDebugger**, encontraremos el mensaje (**Message**).

Finalmente, por medio del método **trace()** de la clase **MonsterDebugger**, depuramos el contenido que corresponda, indicando como primer parámetro el target, y luego, el mensaje de salida. Si bien en este pequeño ejemplo estamos haciendo un output de una cadena de texto, podemos depurar cualquier tipo de elemento: arrays, objetos, estructuras XML, números, etcétera.

Por lo que hemos visto, la implementación de FlashTracer es más sencilla que la de MonsterDebugger. Incluso, con FlashTracer no necesitamos utilizar clases y métodos para generar mensajes de salida; simplemente, con la función **trace()** de Flash podemos visualizar los contenidos. MonsterDebugger es una mejor plataforma para debugging ya que brinda información más detallada y específica sobre los mensajes de salida, frente a FlashTracer, que los imprime como texto plano.

Veamos el siguiente ejemplo. No nos interesa adentrarnos en la sintaxis de ActionScript 3.0 (por lo menos por ahora); simplemente, lo que hacemos es proceder a realizar la carga de una estructura XML y utilizar MonsterDebugger para imprimirla:

```
import flash.net.URLRequest;
import flash.net.URLLoader;
import flash.events.Event;
import flash.net.drm.VoucherAccessInfo;

import nl.demonsters.debugger.MonsterDebugger;

var myInstance:MonsterDebugger = new MonsterDebugger(this);
var request:URLRequest = new URLRequest("http://blog.theflashblog.
    com/?feed=rss2");
var loader:URLLoader = new URLLoader();
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete, false, 0, true);

function onComplete(e:Event):void{
    var xml:XML = new XML(e.target.data);
    trace(xml);
    MonsterDebugger.trace(this, xml);
}
```

Al completarse la carga de la estructura XML, podemos imprimirla de dos maneras: por medio del tradicional **trace()** o usando la clase **MonsterDebugger**. Si observamos el modo en el cual se genera el **output** en FlashTracer, notaremos que es texto plano. Ahora bien, si accedemos a la interfaz de MonsterDebugger, nos encontramos el mensaje de salida correspondiente. Al hacer doble clic sobre él, veremos que tenemos la estructura XML de una manera mucho más organizada, jerárquica y en forma de árbol, lo que nos permitirá realizar una búsqueda mucho más sencilla de los contenidos que necesitemos.

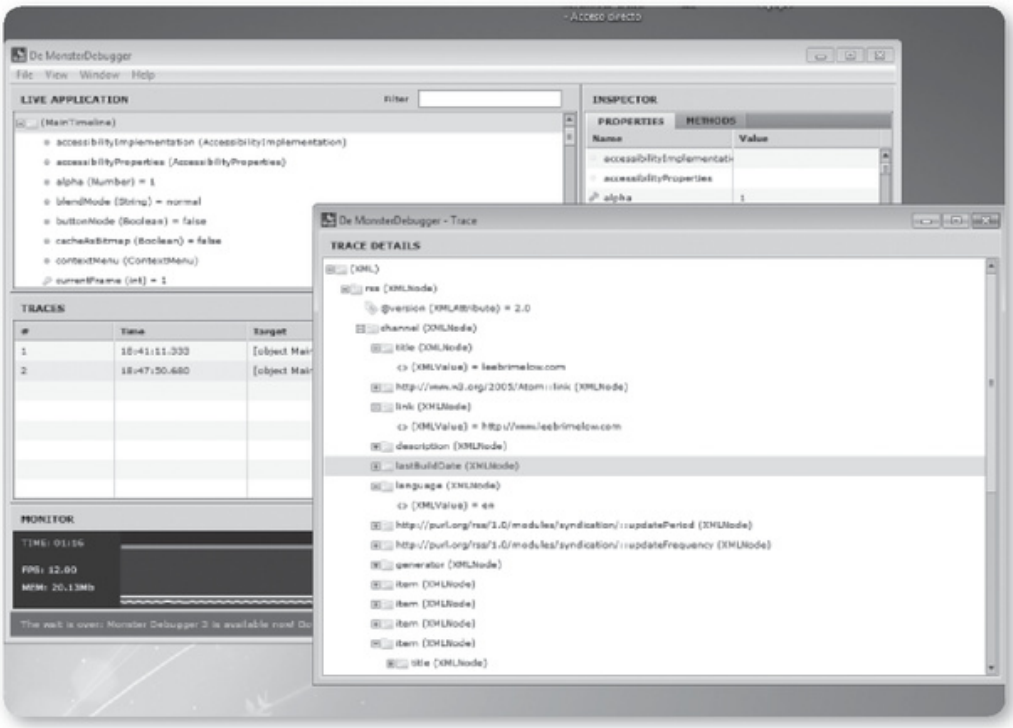



**Figura 7.** Si hacemos doble clic sobre el mensaje de salida que se generó, accedemos a información mucho más detallada.

## ANÁLISIS DE OTROS FRAMEWORKS

Antes de adentrarnos en el desarrollo de un espacio de trabajo, es una buena práctica descargar otros frameworks y analizarlos para conocer sus características y saber de qué manera se articulan sus contenidos. Existen varios frameworks para la Plataforma de Adobe Flash.

## Otra alternativa: Arthropod

**Arthropod** es otra opción a la hora de hacer debug de contenidos. Para emplearlo, debemos descargarlo visitando el sitio web que encontramos en la dirección **http://arthropod.stopp.se**.

Al igual que MonsterDebugger, antes de utilizarlo, debemos importar la clase (en este caso, se trata de **Debug.as**):

```
import com.carlcalderon.arthropod.*;
Debug.log(xml);
```

## Debug con Firebug

Firebug es, probablemente, el add-on más utilizado a la hora de hacer debugging de contenidos. La ventaja que nos brinda depurar código desde él es que no necesitamos tener otras ventanas abiertas además de nuestro navegador, y muchas veces resulta más cómodo leer los mensajes de salida desde él mismo.

## Entre tantas formas de hacer debug, ¿con cuál nos quedamos?

En definitiva, si bien muchas veces es cuestión de gustos, FlashTracer es una opción viable cuando trabajamos en pequeños proyectos con información sencilla, y no necesitamos un debug avanzado y detallado de los contenidos. Por su parte, MonsterDebugger es mucho más viable cuando nos manejamos con estructuras más complejas, en tanto que el add-on Firebug nos permite trabajar desde la comodidad del navegador sin tener que abrir otras ventanas. ¿Con qué opción nos

### PUREMVC FRAMEWORK

A diferencia de **GAIA**, que está claramente orientado a Flash, PureMVC es un espacio de trabajo destinado a trabajar bajo el patrón **MVC** (**Model View Controler**). Debemos tener en cuenta que ofrece una versión para **ActionScript 2.0** y otra para **ActionScript 3.0**. A su vez, se encuentra disponible para varios lenguajes más: **PHP**, **JavaScript**, **Python**, **Java** y **Objective C**, entre otros.

quedaremos? Con las tres. En los próximos capítulos veremos de qué manera podemos programar una clase en ActionScript 3.0 que maneje los mensajes de salida de Flash de modo tal que nos desentendamos de este problema puntual, y sepamos que, elijamos la opción que elijamos, podremos leer dichos mensajes.

# Clases y contenido externo de nuestro framework

En los próximos capítulos crearemos las clases que dan forma a nuestro espacio de trabajo, pero hay algunos paquetes ya desarrollados que nos serán de gran ayuda y resultarán un complemento de nuestro desarrollo. Es una buena idea contar con ellos de antemano.

> MÁS ADELANTE CREAREMOS LAS CLASES PARA NUESTRO ESPACIO DE TRABAJO

Todo el contenido que se nombra a continuación ya se encuentra descargado y listo para utilizar. Para disponer de él, es necesario generar la descarga desde los archivos que acompañan a este libro. De todos modos, a continuación se brinda una breve descripción de cada uno de los paquetes por utilizar y el link desde el cual se puede generar la descarga de los contenidos. Contar con estos links puede ser de gran utilidad cuando el material de los paquetes que acompañan a este libro pase a ser obsoleto o requiera actualizaciones.

## as3CoreLib

Se trata de un paquete de clases de suma utilidad.

as3CoreLib incluye clases para encriptar contenido en MD5 y SHA1; *encoders* para imágenes (**JPG**, **PNG**); y utilidades para manejo de cadenas de texto, números, fechas, etcétera. Se puede descargar desde la dirección web **https://github.com/mikechambers/as3corelib**.

El contenido debe copiarse dentro de la carpeta **com**, la cual puede encontrarse dentro de la carpeta denominada **src**.

## greenSock (TweenLite, TweenMax, etc.)

Seguramente, la mayoría de los lectores ya estén familiarizados con este paquete. Contiene las clases **TweenLite** y **TweenMax**, entre otras, mediante las cuales podemos generar **tweenings** de una forma sencilla e intuitiva. Existen otros motores de animación, pero este será el que utilizaremos en nuestro desarrollo. Podemos descargarlo desde **www.greensock.com**. El contenido de este paquete debe copiarse dentro de la carpeta **com**, que se encuentra dentro de la carpeta **src**.

## asual (SWFAddress)

Necesitamos este paquete para hacer uso de **SWFAddress**, una clase que nos permite manejar el **deeplinking** del navegador dentro de nuestros desarrollos en Flash. Haremos uso de este paquete en el **Capítulo 6** de este libro. Encontramos la descarga en el sitio web que está en la dirección **www.asual.com/swfaddress**.

El contenido de este paquete debe copiarse dentro de la carpeta **com**, que se encuentra dentro de la carpeta denominada **src**.

## SWFObject

Es importante saber que, si utilizamos FlashDevelop, el programa mismo crea estos archivos al hacer un nuevo proyecto. Pero en caso de que decidamos utilizar otro editor de código o el mismo Flash CS5, precisaremos descargar **SWFObject**, que nos permitirá embeber contenido Flash dentro del navegador de forma transparente al emplear JavaScript para hacerlo. En el **Capítulo 4** veremos de qué manera podemos utilizar SWFObject para saber cuándo crear contenido alternativo para aquellos dispositivos que estén accediendo a nuestro sitio y no tengan soporte para Flash. Podemos descargarlo desde la dirección web **http://code.google.com/p/swfobject**.

El contenido que descargamos del sitio se encuentra dentro de una carpeta denominada **js**; debemos moverla dentro de nuestra carpeta **bin**.

## libSpark.as

Esta sencilla clase nos permite brindarles la posibilidad de usar la rueda del mouse a los usuarios de Mac. Una vez que la descargamos,

debemos agregarla a la carpeta **src/org/libspark/ui**. Puede obtenerse desde el sitio web que se encuentra en la dirección **www.libspark.org/browser/as3/SWFWheel/trunk/src/org/libspark/ui?rev=2297**.

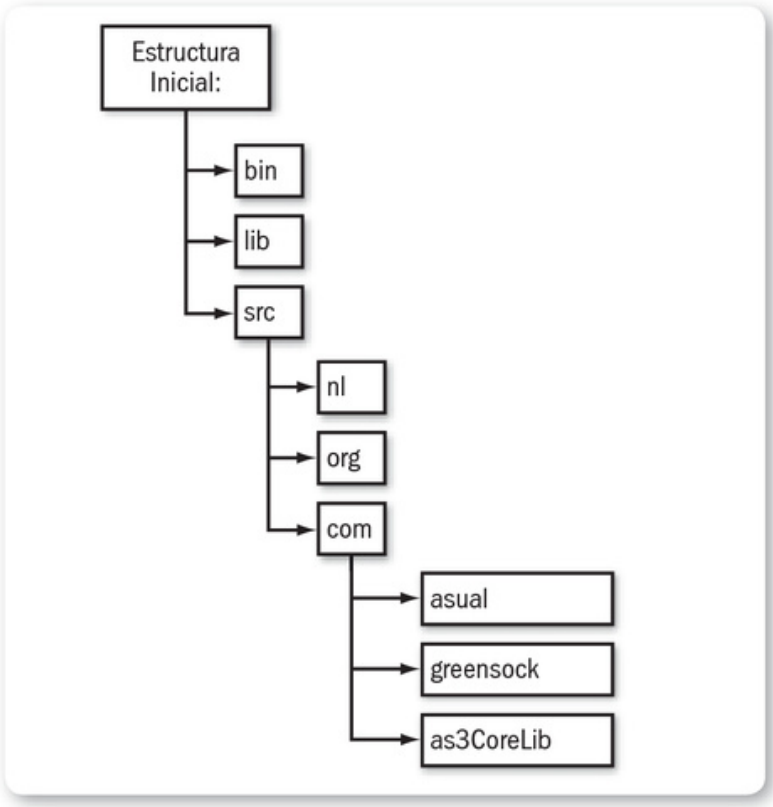



**Figura 8.** Una vez concluida la descarga de los paquetes, esta es la estructura con la que debemos contar.

## RESUMEN

Concluido el segundo capítulo de este libro, ya contamos con la estructura básica que necesitamos para encarar nuestro entorno de desarrollo. Lo más importante de esta sección es saber por qué resulta importante trabajar de manera estructurada y cuáles son las ventajas que surgen de hacerlo. A su vez, hemos descargado todas las utilidades (add-ons, clases, plugins) que nos ayudarán tanto para las actividades planteadas en este libro como para cualquier otro desarrollo.

# Actividades

## TEST DE AUTOEVALUACIÓN

1. ¿Cuál es la ventaja de contar con un espacio de trabajo?
2. ¿En qué caso y para qué tipo de desarrollo es ideal tener uno?
3. ¿Cuáles son sus principales beneficios?
4. ¿Cuáles son sus limitaciones?
5. ¿Para qué necesitamos hacer *debugging* de contenidos?
6. ¿Con qué herramientas contamos para hacerlo?
7. ¿Por qué utilizaremos FlashDevelop en lugar de la IDE de Flash CS5 para programar?
8. ¿Por qué es importante separar nuestro código fuente del desarrollo?
9. ¿Qué archivos van dentro de la carpeta **bin**?
10. ¿Cuáles van dentro de **src** y cuáles dentro de **lib**?

## ACTIVIDADES PRÁCTICAS

1. Instale un servidor local.
2. Cree un nuevo proyecto con FlashDevelop.
3. Coloque algunos archivos **FLA**, **SWF** y **AS** en cada una de las carpetas que les corresponda.
4. Exporte el contenido y genere mensajes de salida a través del uso de MonsterDebugger.
5. Pruebe distintas maneras de depurar el código.

*03

# Reutilización y escalabilidad

Una vez que contamos con la estructura básica de nuestro proyecto, veremos de qué manera comenzar a organizar los paquetes para que su contenido sea reutilizable y escalable, y mantener un orden y una organización dentro del trabajo que nos permita manejarnos en forma prolija.

# Importancia en la organización de un proyecto

En el **Capítulo 2** de este libro, nombramos los cuatro pilares que sustentan la existencia de un espacio de trabajo. Ahora bien, para poder garantizarlo, debemos focalizar nuestro desempeño a fin de que se den dos condiciones dentro de él:

- Su contenido debe ser escalable.
- Su contenido debe ser reutilizable.

Si destinamos nuestros esfuerzos a un desarrollo que no nos permita volver a utilizarlo o no nos deje hacerlo crecer en el futuro, en gran medida, estaremos desperdiciando nuestro tiempo. Estos son los dos puntos vitales que debemos tener en mente cada vez que pensamos en un desarrollo y en su futura expansión, y para lograr un verdadero entorno en el cual se den ambas condiciones, es imprescindible **mantener un orden** y saber:

- Qué clases pertenecen a nuestro framework orientado a la Web.
- Qué clases pertenecen a nuestra librería personal.
- Qué clases son propias y específicas, y por lo tanto corresponden al desarrollo que se está llevando a cabo.

Como hemos visto, la carpeta **bin** contiene los archivos finales de nuestro proyecto (archivos de JavaScript, PHP, XML, imágenes, audios, películas **.SWF**), la carpeta **lib** contiene los archivos **.FLA** del desarrollo, y la carpeta **src**, los paquetes con las clases por utilizar.

Es hora de comenzar a crear los paquetes que contendrán nuestras clases. Para esta ocasión llamaremos a estos paquetes de la siguiente forma: **myPackages**, **project** y **usersFramework**.

## NOMBRES DE PAQUETES Y CLASES

A modo de convención, los nombres de los paquetes que contienen clases llevan letra capital minúscula, mientras que los nombres de las clases comienzan con mayúscula. Mantener estas convenciones nos permite ser más prolijos dentro de nuestro entorno. Ejemplo: **com.myPackage.utils.MyClass**.

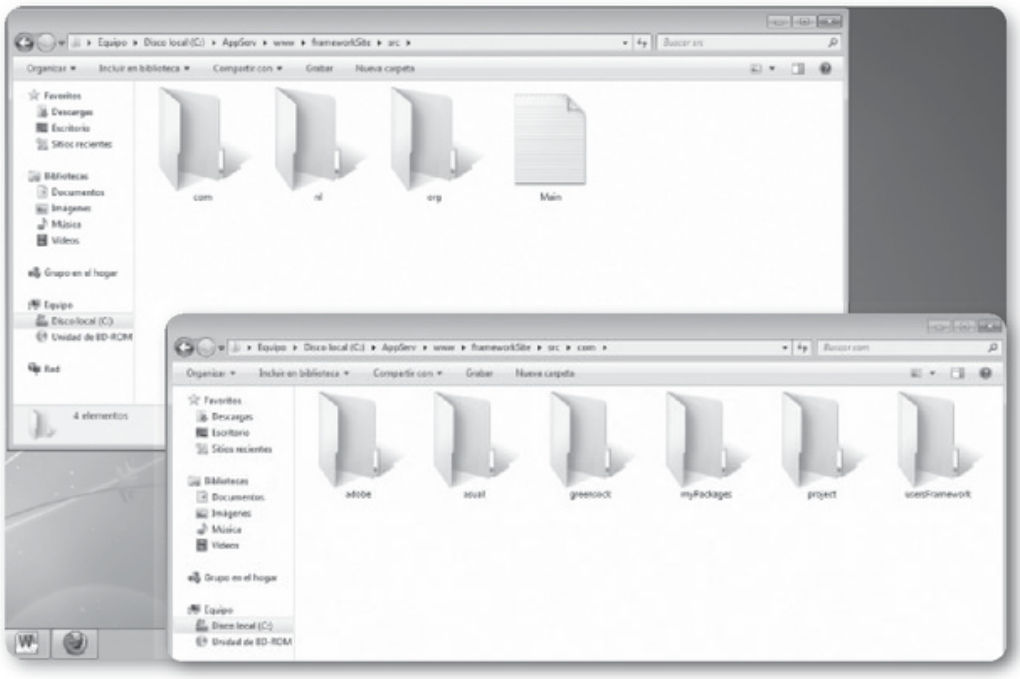



**Figura 1.** En esta imagen vemos la estructura actual de nuestra carpeta **src**. Dedicaremos la mayor parte de este capítulo a crear nuevos paquetes y clases dentro de dicho directorio.

Es imprescindible que diferenciemos estos tres paquetes y sepamos de antemano cuál es la función de cada uno de ellos. Cuando concluyamos con este libro y contemos con un framework escalable y reutilizable, se deberá, principalmente, al hecho de mantener esta estructura que acabamos de generar, la cual tiene funciones bien definidas y distintas entre sí, que es importantísimo conocer antes de continuar.

Seguramente, quienes cuenten con un **background** de desarrollo ya estarán familiarizados con este tipo de estructura, pero puede resultar confuso para quienes estén buscando un desarrollo más estructurado.

## ORGANIZACIÓN DE LOS CONTENIDOS

No es en vano que hayamos dedicado tres capítulos a la organización de los contenidos e insistamos en la importancia de mantener un orden. Hacer esto es imprescindible a fin de optimizar nuestro tiempo de desarrollo, poder escalar nuestras clases, lograr mejores implementaciones y reutilizar el código.

# Paquete myPackages

Como podemos deducir de su nombre, se trata de un paquete que se encargará de contener nuestros propios paquetes.

El contenido de este paquete resulta de la necesidad de disponer de contenido reutilizable y que no tiene que ver con las especificidades del trabajo puntual que estemos desarrollando. Pensémoslo con un ejemplo. Este paquete sería el lugar indicado para una clase que generara valores aleatorios. ¿Por qué generalmente uno cuenta con una clase para este tipo de contenido? Por el simple hecho de que son funciones comunes a cualquier tipo de desarrollo. La necesidad de crear un valor aleatorio puede surgirnos durante el desarrollo de un sitio web, de un videojuego, de una aplicación para un dispositivo móvil, de una de escritorio, y de prácticamente cualquier tipo de desarrollo. Es decir, es una función común si consideramos el amplio espectro de posibilidades con que cuenta la clase: por este motivo, la incluimos dentro de nuestro propio paquete, el cual utilizaremos cada vez que iniciemos cualquier proyecto, independientemente del tipo de desarrollo que sea.

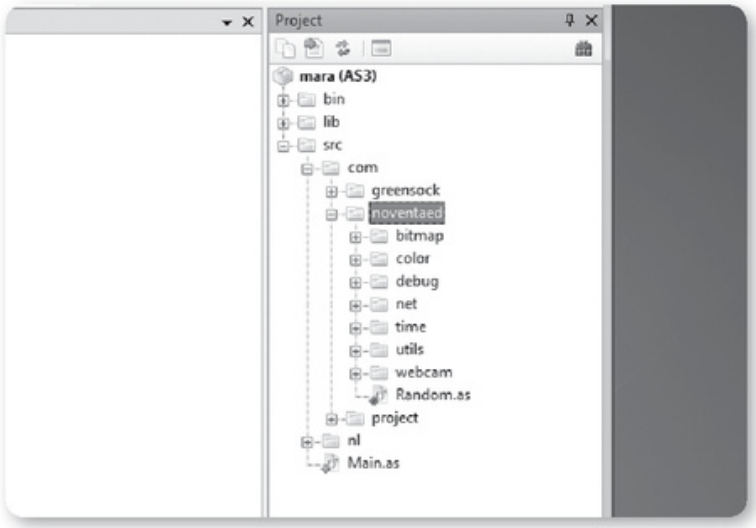



**Figura 2.** Estructura implementada por **#90ED** vista desde **FlashDevelop**.

Debemos tener en cuenta que este paquete también sería el lugar adecuado para agregar una clase que se ocupara de cargar una estructura XML, o bien una que manejara correctamente el debugging de los contenidos de nuestras películas Flash. Recordemos: estas son acciones que, seguramente, se requieran ante cualquier tipo de desarrollo y no son específicas de ninguno en particular.

# Paquete project

El paquete **Project** es todo lo contrario de **myPackages**. Mientras que este último contiene clases para resolver problemas comunes a cualquier desarrollo, **project** contiene las clases específicas de un proyecto en particular. Es decir, son clases que resultan útiles solamente para el proyecto de turno, y no para otro. En el **Capítulo 1** de este libro mencionamos que una de las grandes ventajas de contar con un espacio de trabajo es que nos permite dedicarnos de lleno a lo específico de un desarrollo. Pues bien, esta carpeta es la que lo contiene. Es la que incluirá las clases que solamente resultarán útiles y funcionales para un proyecto determinado, y no para los demás. Si se tratase de un sitio web, este directorio contendría las clases de cada una de las páginas de dicho sitio. Si fuese un juego, contendría todo el código que lo hará funcionar.

# Paquete usersFramework

Dentro del paquete **usersFramework** agregaremos todas las clases que son comunes a un tipo de desarrollo en particular: **el desarrollo de sitios web**. Tal como hemos mencionado en el **Capítulo 2**, este libro se centra en la creación de un espacio de trabajo basado en sitios web. Las clases dentro de este paquete se encargarán de manejar la funcionalidad básica de cualquier sitio:

- La navegación de los contenidos.
- La carga y descarga de películas.
- El preloader del sitio.
- El manejo del deeplinking.
- El acceso y la modificación de la información vía XML.
- El menú del sitio y el menú contextual.

Debemos considerarlo el **motor** de nuestro desarrollo.

## PAQUETE DE BIGSPACESHIP

El paquete que hemos denominado **myPackages** lleva por nombre el de la empresa que lo desarrolló. La compañía **BigSpaceship** puso a disposición del público parte del paquete con el cual trabaja. El contenido puede descargarse de **www.bigspaceship.com/blog/labs/bss-actionscript-package.**

# XML para crear un sitio en Flash

La información que vamos a utilizar para crear nuestro sitio se la pasaremos a Flash por medio de una estructura XML. Con XML es sumamente sencillo modificar el contenido de forma externa en caso de tener que hacerlo; no es necesario tocar código Flash si precisamos hacer alguna modificación y, a la vez, podemos utilizar PHP para parsear la estructura XML y crear contenido alternativo en aquellos casos en que un *smartphone* o una computadora no soporte Flash.


**Figura 3.** Estructura XML que utilizaremos para indicar la información con la que tiene que contar nuestro desarrollo. De esta manera es importante diferenciar el nodo **site** del nodo **page**.

Debemos tener en cuenta que el XML que vamos a utilizar se encuentra dividido en dos nodos: por un lado, encontramos **site**, y por otro, **page**. El nodo **site** tiene atributos que contienen la información referente al sitio en sí, y cada nodo **page** tiene toda la información referida a cada una de las páginas que contendrá el sitio.

Nodo **site**:

```
<site title="el titulo del
sitio" menu="true" history="true" delimiter="-" accessibility="true"
preloader="swf/preloader.
swf" framerate="40" fullScreen="true" defaultPage="index">
```

Nodo **page**:

```
<page id="index" urlFriendly="index" src="swf/index.
swf" title="index" menu="true">
```

# Nodo site

Veamos qué atributos podemos utilizar dentro del nodo **site** y qué valores definir para cada uno de ellos:

```
<site title="el titulo del sitio" menu="true" history="true" delimiter="-" accessibility="true"
preloader="swf/preloader.swf" framerate="40" fullScreen="true" landingPage="index">
```

**title**:
El nombre del sitio web.

**Atributo**:

```
<site title="el titulo del sitio"
```

## ALCANCE DE UN ESPACIO DE TRABAJO

Como ya hemos mencionado anteriormente, un espacio de trabajo no abarca la totalidad de las soluciones ante determinado tipo de desarrollo. Por el contrario, un framework es una herramienta con la que contamos que tiende a ser de utilidad en la mayoría de los situaciones.

**Valores aceptados**:
Una cadena de texto que contenga el nombre del sitio web.

**Valor por defecto**:
Ninguno.

**Tipo de valor**:
Obligatorio.

## menu

Sabemos que Flash cuenta con un menú contextual, que puede sernos de gran utilidad mientras trabajamos en un desarrollo web. Por medio de este atributo, es posible acceder a indicar si nuestro sitio contará con un menú contextual o no.

**Atributo**:

```
<site menu="true"
```

**Valores aceptados**:
True/False

**Valor por defecto**:
True.

**Tipo de valor**:
Opcional.

### CONTENIDO EXTERNO

Es necesario tener en cuenta que la gran ventaja de mantener la información que genera un entorno de trabajo fuera de él es que, en caso de querer llevar a cabo las modificaciones que son requeridas, no será preciso modificar el código fuente del desarrollo: simplemente, editando el archivo **.XML**, de esta forma es posible indicar cambios en el comportamiento de nuestro sitio web.


**Figura 4.** Es posible brindarle al usuario acceso a las secciones de nuestro menú a través del menú contextual que ofrece Flash.

## history

Implementando **SWFAddress** en Flash, podemos manejar el **deeplinking** del sitio (manejo en profundidad de los contenidos a través de los botones de navegación del navegador). A su vez, los browsers cuentan con la opción de facilitarnos el historial de navegación, por medio de la cual podemos definir que esta aparezca o no. Le daremos mucha importancia al deeplinking y al historial de navegación dentro de nuestro entorno de trabajo. En los próximos capítulos veremos de qué manera implementaremos estas soluciones para mejorar la navegación

**Atributo**:

```
<site history="true"
```

**Valores aceptados**:
True/False

**Valor por defecto**:
True.

**Tipo de valor**:
Opcional.

## delimiter

Se utiliza para indicar un delimitador entre el nombre del sitio y la sección en la cual nos encontramos.

**Atributo**:

```
<site history="true"
```

**Valores aceptados**:
Cualquier cadena de texto con no más de dos caracteres.

**Valor por defecto**:
"-"

**Tipo de valor**:
Opcional.

## accessibility

Flash ofrece varias utilidades en lo que respecta a la accesibilidad. Por medio del uso de este parámetro, es posible indicarle a nuestro desarrollo que tenga en cuenta dichas utilidades y que de esta forma se encargue de aplicar criterios de accesibilidad al sitio.

**Atributo**:

```
<site accessibility="true"
```

**Valores aceptados**:
True/False

**Valor por defecto**:
True.

**Tipo de valor**:
Opcional.

## preloader

A través de este atributo, es posible indicarle la ruta de un archivo **.SWF** que será el **preloader** utilizado para la carga de las páginas. Tener el preloader en una película separada del resto nos permite trabajarlo de forma independiente, sin mezclarlo con el contenido específico de cada página.

**Atributo**:

```
<site preloader="swf/preloader.swf"
```

**Valores aceptados**:
Ruta de un archivo .SWF.

**Valor por defecto**:
Ninguno.

**Tipo de valor**:
Opcional.

## framerate

Por medio de ActionScript 3.0, podemos modificar el *framerate* (fotogramas por segundo) de una película. La posibilidad de manejar esta información de manera externa nos evita tener que abrir el archivo **.FLA** para editarla en caso de que necesitemos modificar algo.

**Atributo**:

```
<site framerate="40"
```

**Valores aceptados**:
Valores numéricos entre 12 y 60.

**Valor por defecto**:
24

**Tipo de valor**:
Opcional.

## fullscreen

Podemos indicarle a nuestra película que cuente con la posibilidad de reproducirse en modo normal o fullscreen. Este parámetro también se maneja de manera externa. Veremos de qué forma implementar esta utilidad en el menú contextual de nuestro desarrollo para poder modificar la visualización de los contenidos desde él.

**Atributo**:

```
<site fullscreen="true"
```

**Valores aceptados**:
True/False.

**Valor por defecto**:
False.

**Tipo de valor**:
Opcional.

## landingPage

Este atributo indicar a la película principal cuál es la primera página que se debe abrir al inicializarse el sitio. Si no tiene un valor, abriremos la página indicada en el primer nodo **page** de la estructura XML.

**Atributo**:

```
<site landingPage="index"
```

**Valores aceptados**:
Cadena de texto.

**Valor por defecto**:
Ninguno.

**Tipo de valor**:
Opcional.

# Nodos page

Mientras que dentro del nodo **site** indicamos valores para los atributos que son comunes al sitio en sí, dentro de cada nodo **page** debemos indicar la información de cada una de las páginas que contendrá el sitio. El nodo **page** presenta menos atributos que **site**. Veamos cuáles se pueden utilizar:

```
<page id="index" urlFriendly = "index" src="swf/index.swf" title = "index"
menu="true">
```

## id

Para cada página necesitamos un valor que nos permita diferenciarla del resto. Para esto, utilizaremos el atributo id.

**Atributo**:

```
<site id="index"
```

**Valores aceptados**:
Cadena de texto indicando el nombre de la página.

### NUEVOS ATRIBUTOS

Una de las ventajas de trabajar de esta manera es que, en caso de ir descubriendo nuevas funcionalidades, es fácil añadirlas el entorno de trabajo y manejarlas de manera externa. En breve veremos de qué modo podemos acceder a esta información desde Flash para, luego, asignarle su verdadera funcionalidad.

**Valor por defecto**:
Ninguno.

**Tipo de valor**:
Obligatorio.

### src

Utilizamos este atributo para indicar la ruta de la película correspondiente a la página.

**Atributo**:

```
<site src="swf/index.swf"
```

**Valores aceptados**:
Cadena de texto indicando la ruta del archivo.

**Valor por defecto**:
Ninguno.

**Tipo de valor**:
Obligatorio.

## urlFrienfly

Necesitamos una cadena de texto para utilizar el **deeplinking** del sitio. Este es el valor que aparecerá en la barra del navegador. A su vez, esta opción nos permite acceder directamente desde el browser a una sección específica del sitio. En el capítulo final de este libro veremos de qué manera podemos utilizar el deeplinking para acceder a una contenido específico dentro de una sección específica del sitio.

**Atributo**:

```
<site urlFriendly="index"
```

**Valores aceptados**:
Cadena de texto sin espacios.

**Valor por defecto**:
Ninguno.

**Tipo de valor**:
Obligatorio.


**Figura 5.** El valor que indiquemos en el atributo `urlFriendly` es el que aparecerá en la barra del navegador.

## title

Así como tenemos un título común al sitio, es normal que cada página cuente con su nombre de sección. Este valor se utiliza para concatenarlo al nombre final del sitio web.

El título se conforma como vemos a continuación:

### PRECAUCIONES

El hecho de que un framework cumpla con una serie de tareas repetitivas puede generar miedo en vistas a que todos los desarrollos que hagamos sean iguales o similares entre sí. Es por este mismo motivo que el desarrollo que llevaremos a cabo no tendrá repercusión en el aspecto visual del trabajo. Simplemente lo limitaremos a las tareas específicas que no tengan repercusión en el armado de los contenidos.

**nombre del sitio web + delimitador + nombre de página**

**Atributo**:

```
<site title="index"
```

**Valores aceptados**:
Cadena de texto indicando la ruta del archivo.

**Valor por defecto**:
Ninguno.

**Tipo de valor**:
Opcional.

## menu

Si bien el sitio cuenta con un menú, tal vez queramos que en una sección específica este no sea visible a los usuarios que visiten nuestro desarrollo. Para estos casos, cada nodo de página dispone del atributo **menu**. Si le definimos como valor **false**, el menú contextual no aparecerá cuando nos encontremos en dicha página.

**Atributo**:

```
<site menu="true"
```

### ALCANCE DE ESTE ESPACIO DE TRABAJO

Es importante mentalizarse de que un framework no es más que una herramienta con la cual contamos para solucionar un problema específico. Ahora bien, es muy probable que esta solución no sea la indicada para todos los casos: probablemente, existan sitios que no se adapten a esta estructura, y entonces tengamos que pensar en otra alternativa. Lo que se busca con este tipo de desarrollo es resolver problemas comunes a situaciones de índole similar, pero ningún framework soluciona absolutamente todos los inconvenientes.

**Valores aceptados**:
True/False.

**Valor por defecto**:
True.

**Tipo de valor**:
Opcional.

# Archivos .XML, .FLA, .SWF y .AS

Solamente falta un paso antes de dar comienzo a la programación de los contenidos de nuestro espacio de trabajo. Insistimos en reiteradas oportunidades respecto a la importancia de mantener un orden y una coherencia en la organización de los archivos que componen el espacio de trabajo. Veamos de qué manera ubicarlos.

## Estructura XML

Ya mencionamos anteriormente que todos los archivos relacionados con el resultado final de nuestro desarrollo irán en la carpeta denominada **bin**. Ahora bien, dentro de ella, también debemos mantener un orden. En un proyecto pequeño, quizá no nos traiga mayores problemas tener los archivos desorganizados, pero en cualquier desarrollo que implique una importante cantidad de archivos, es conveniente conservar una estructura y respetarla.

### PROGRAMACIÓN ORIENTADA A OBJETOS

En el **Capítulo 5** de este libro nos encargaremos de trabajar en la programación de nuestro sistema. Veremos de qué manera la Programación Orientada a Objetos nos brinda una serie de ventajas y soluciones cuando necesitamos crear clases reutilizables y llevar a cabo implementaciones.

Dentro de la carpeta **bin** será necesario que realicemos la creación de otra llamada **xml**, y dentro de esta última, colocaremos la estructura XML que da forma a nuestro sitio. Lo llamaremos **site.xml**.

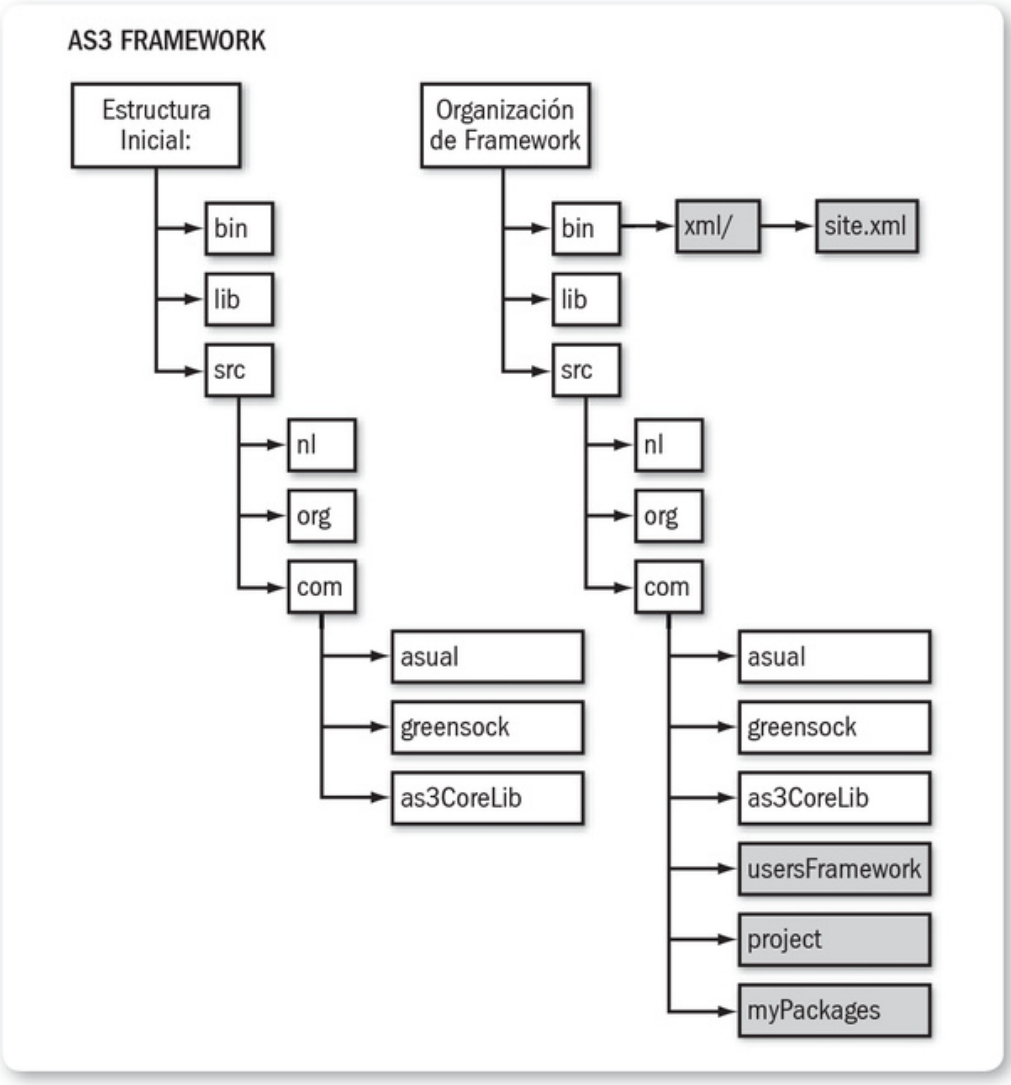



**Figura 6.** Estos son los cambios que hemos introducido a lo largo de este capítulo respecto a la estructura del capítulo anterior.

# Archivos .FLA

Ya hemos dicho que los archivos del tipo **.FLA** deberán ubicarse dentro de la carpeta **lib** de nuestra estructura. En primer lugar, debemos diferenciar dos grupos de archivos de este tipo:

- Por un lado, se encuentra el archivo **.FLA** principal de la aplicación.
- Por el otro, los archivos **.FLA** de cada una de las páginas que contenga el proyecto (a modo de analogía, podemos recordar la estructura XML que hemos ido creando páginas atrás: todo lo referente al sitio –nodo **site**– corresponde a la página principal –la llamaremos **main.fla**–, y por cada página –nodo **page**– tendremos su correspondiente archivo **.FLA** de página).

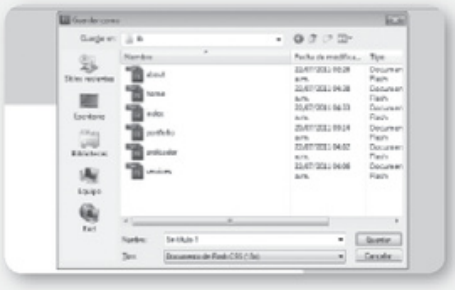
Si bien esto no es un problema, en ambos casos se nos presentan dos situaciones para resolver:

- Las clases que vamos a utilizar dentro de los archivos **.FLA** están en otro directorio (recordemos que para los códigos fuente utilizaremos la carpeta **src**).
- Los contenidos deben exportarse a la carpeta **bin**, y los archivos **.FLA** se ubican en la carpeta **lib**.

Por estos motivos, debemos hacer dos pequeños cambios en la configuración de nuestros archivos.

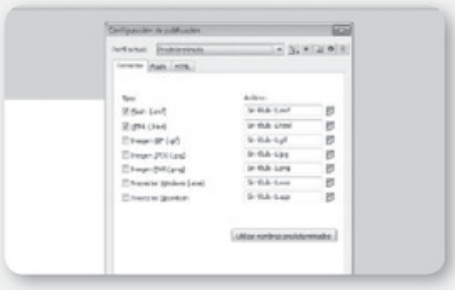
## EXPORTAR CONTENIDO A OTRO DIRECTORIO

**01**



Cree un nuevo archivo de Flash y guárdelo dentro de la carpeta **lib**.

**02**



A continuación, acceda a **Configuración de Publicación**. Tiene dos maneras de hacerlo: por medio del atajo **CTRL+SHIFT+F12**, o bien desde el menú denominado **Archivo/Configuración de publicación...**.

**03**

Deberá modificar la ruta en la cual publica su archivo **.SWF**. Para esto, dentro de la solapa **Formato**, haga clic en la carpeta que se encuentra junto al nombre del archivo **.SWF**, y navegue hasta el directorio **bin/swf**.

De esta manera, luego de efectuar los pasos ya mencionados ya tenemos nuestro archivo **.FLA** en un directorio distinto del **.SWF**. Lo que nos queda por hacer, es asignarle su respectiva clase.

# Archivos .AS

Es importante recordar que diferenciamos un archivo **.FLA** (encargado de crear la película principal de nuestro desarrollo, **main.fla**) y varios archivos **.FLA** que conforman cada una de las páginas del sitio (**sound.fla**, **video.fla**, **images.fla**, etcétera). Al archivo **main.fla** debemos asignarle como clase lo que llamaremos **clase principal** (o **Main Class**), que dará inicio a todo el desarrollo.

## Main.as

Esta clase se crea por defecto al generar un nuevo proyecto en FlashDevelop. Se coloca en el primer nivel del directorio **src**.

Debemos tener en cuenta que desde esta clase nos encargaremos de iniciar el espacio de trabajo que vamos a desarrollar , por esta razón, aunque no debemos crearla, es importante conocerla.

### USO DE CLASES

Interpretamos que todo lector que se acerque a esta obra cuenta con conocimientos sobre el manejo de clases en ActionScript 3.0 y se siente cómodo con ellas. Si bien analizaremos el contenido de las mismas en los casos en los que se lo requiera, es de esperar que el lector se encuentre familiarizado con este tipo de desarrollo, de esta forma el aprendizaje será más fluido.

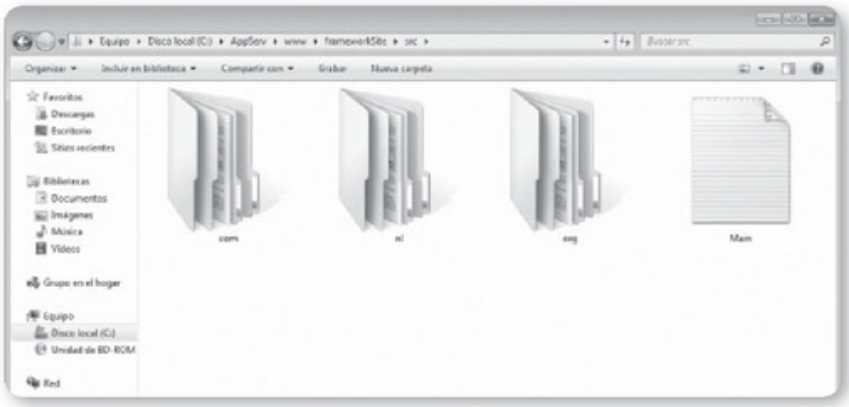


**Figura 7.** En esta imagen podemos ver la clase **`Main.as`**, por fuera del resto de los paquetes disponibles.

## Archivos .AS de páginas

Cada página de nuestro sitio necesita de una clase principal. A diferencia de la clase **Main.as**, la cual se coloca en el primer nivel del paquete **src**, a las clases principales de cada página las pondremos en otro lugar que nos permita mantener un mejor orden. Con este fin, es importante recordar lo que mencionamos al comenzar este capítulo. ¿Qué paquete es el que contiene todas las clases propias del desarrollo? Como sabemos, se trata del paquete **project**.

Dentro de él, entonces, debemos encargarnos de crear un nuevo directorio llamado **pages**, y en él guardaremos todas las clases principales de cada una de las secciones del sitio.

De esta manera, sabremos que, cada vez que creemos una nueva sección de nuestro sitio, deberemos:

- Crear un nodo **page** dentro de la estructura XML y posteriormete asignarle los datos de dicha página.

### CONTENIDO DEL LIBRO

Cabe recordar que todo lo que vamos haciendo a lo largo de estas páginas ya se encuentra realizado y listo para su uso. Solo es necesario descargar los archivos de donde corresponda. Es de gran utilidad contar con ellos, principalmente, para poder seguir los pasos que vamos haciendo durante este capítulo.

- Crear su respectivo archivo **.FLA** y colocarlo dentro del directorio **lib**.
- Crear su respectiva clase y posteriormente proceder a colocarla dentro del directorio **src/project/pages/NombrePage.as**.

Esto es de suma utilidad porque nos evita mezclar el código de nuestro proyecto. Por lo general, las clases principales de una página cumplen una función distinta del resto de las clases de nuestro proyecto (es la que se encarga de inicializar todo). De esta manera, logramos mantener un mejor orden y nos vamos haciendo de un procedimiento: sabemos que cada vez que agreguemos una nueva página, solamente bastará con tres sencillos pasos, y no tendremos que estar mezclando clases con códigos que cumplen distintas funciones.

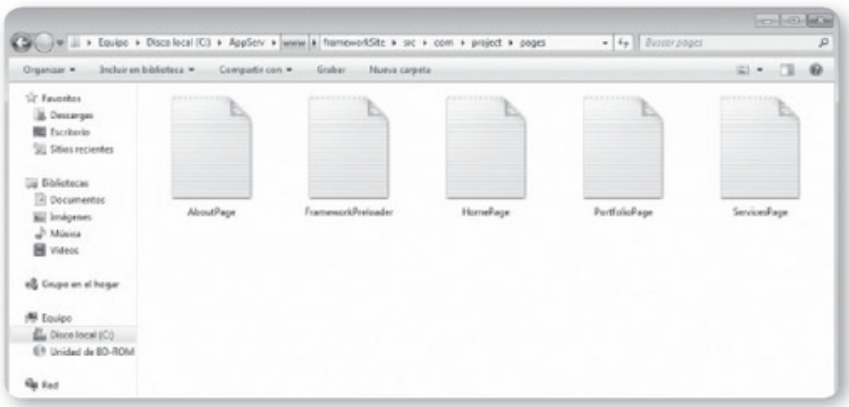



**Figura 8.** Algunos ejemplos de clases dentro del directorio **page**. Una buena idea es escribirlos con su nombre seguido de **Page**.

Nos queda un único paso por realizar antes de comenzar con la programación del contenido. Anteriormente, al crear los archivos **.FLA**, mencionamos dos situaciones por resolver. Por un lado, la necesidad de exportar en otro directorio. Eso lo hemos resuelto en el paso a paso anterior. Ahora nos queda por solucionar nuestro último paso: las clases se encuentran en otro directorio distinto del de nuestro archivo **.FLA**. Por lo tanto, debemos cambiar la ruta de origen que el archivo **.FLA** consulta para obtener su clase principal. Veamos cómo hacerlo.

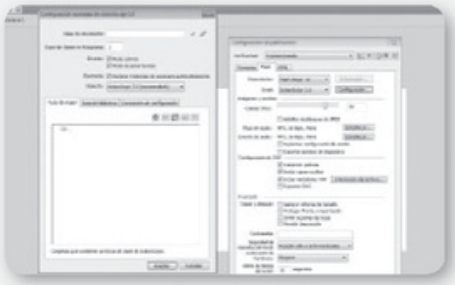
Hasta el momento, hemos creado una estructura XML que contiene la información necesaria para nuestro desarrollo, y hemos colocado los archivos **.FLA**, **.SWF** y **.AS** en distintos directorios.

## VINCULAR .FLA A CLASES EN OTRO DIRECTORIO

**01**

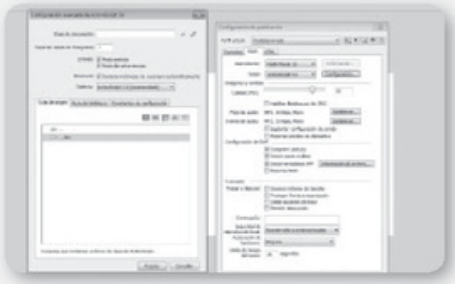
Abra un archivo **.FLA** y acceda al panel **Configuración de publicación** a través del atajo **CTRL+SHIFT+F12**, o bien desde **Archivo/Configuración de publicación...**. Diríjase a la solapa **Flash** y, dentro de ella, presione el botón **Configuración...**. De esta manera, accederá a **Configuración avanzada de ActionScript 3.0**.

**02**



Una vez allí, deberá agregar una nueva ruta de origen. Para hacerlo, en la solapa **Ruta de origen** presione el botón **Añadir nueva ruta** (representado con un signo más). Se habilitará la posibilidad de asignar una nueva ruta.

**03**

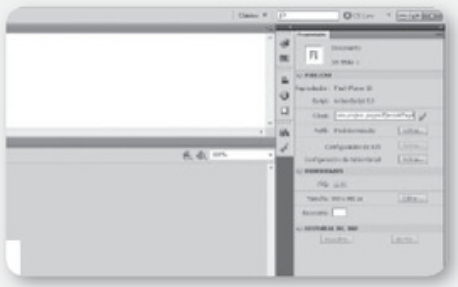


Asigne el nuevo directorio. Considerando que para acceder al directorio **src** desde el directorio **bin** deberá bajar un nivel, simplemente asigne **../src** como nueva ruta de origen.

## ACTIONSCRIPT 3.0

Entre las novedades de ActionScript 3.0 encontramos las siguientes: excepciones y tipos en tiempo de ejecución, clases cerradas y cierres de métodos, expresiones regulares y espacios de nombres. Por otra parte, la versión 2.0 solo ofrecía un tipo numérico mientras que la versión 3.0 posee **int** y **unit**.

04



De esta manera, cuando indique la ruta de la clase, independientemente de que el archivo **.AS** se encuentre en un directorio distinto, el archivo **.FLA** lo localizará, ya que le indicamos que lo busque dentro de la carpeta **src**. Para acceder a la clase que corresponde, indique su ruta: **com.project.pages.EjemploPage**.

Es de vital importancia mantener un orden y una organización. La modalidad de ordenar los contenidos que planteamos en este libro no es la única. Seguramente existan otras maneras y tal vez se adapten mejor a las necesidades de cada desarrollador. Lo importante es conservar un orden, y la gran ventaja de haber invertido el tiempo en esta organización es que, a partir de ahora, podremos mantener nuestros archivos por separado y, en caso de que queramos realizar cambios, sabremos dónde hallar cada uno y tendremos acceso a ellos de forma clara. Quizá no se logre ver las ventajas de este sistema de organización en un desarrollo pequeño, pero a lo largo de este libro iremos haciendo crecer considerablemente este entorno, y entonces notaremos lo ventajoso y provechoso que es tener todos los contenidos por separado. A su vez, mantener una estructura como esta (o similar) es ideal para los grupos de desarrollo multidisciplinarios. Pensemos en un equipo de trabajo: imaginemos un grupo de programadores, uno de diseñadores, y otro llevando a cabo la implementación. Los desarrolladores no tienen necesidad de salir de la carpeta **src**, y pueden centrar sus esfuerzos en ella. Los diseñadores trabajarán en la parte gráfica de los archivos **.FLA**, dentro de la carpeta **lib**, y quienes hagan la implementación del sitio no tendrán por qué conocer qué hay en el resto de las carpetas: solamente necesitarán saber que todos los archivos por subir estarán dentro de la carpeta **bin**. Cuando estemos frente a un gran desarrollo, seguramente recordaremos estas páginas.

> A LO LARGO DE ESTE LIBRO HAREMOS CRECER EL ENTORNO EN FORMA CONSIDERABLE

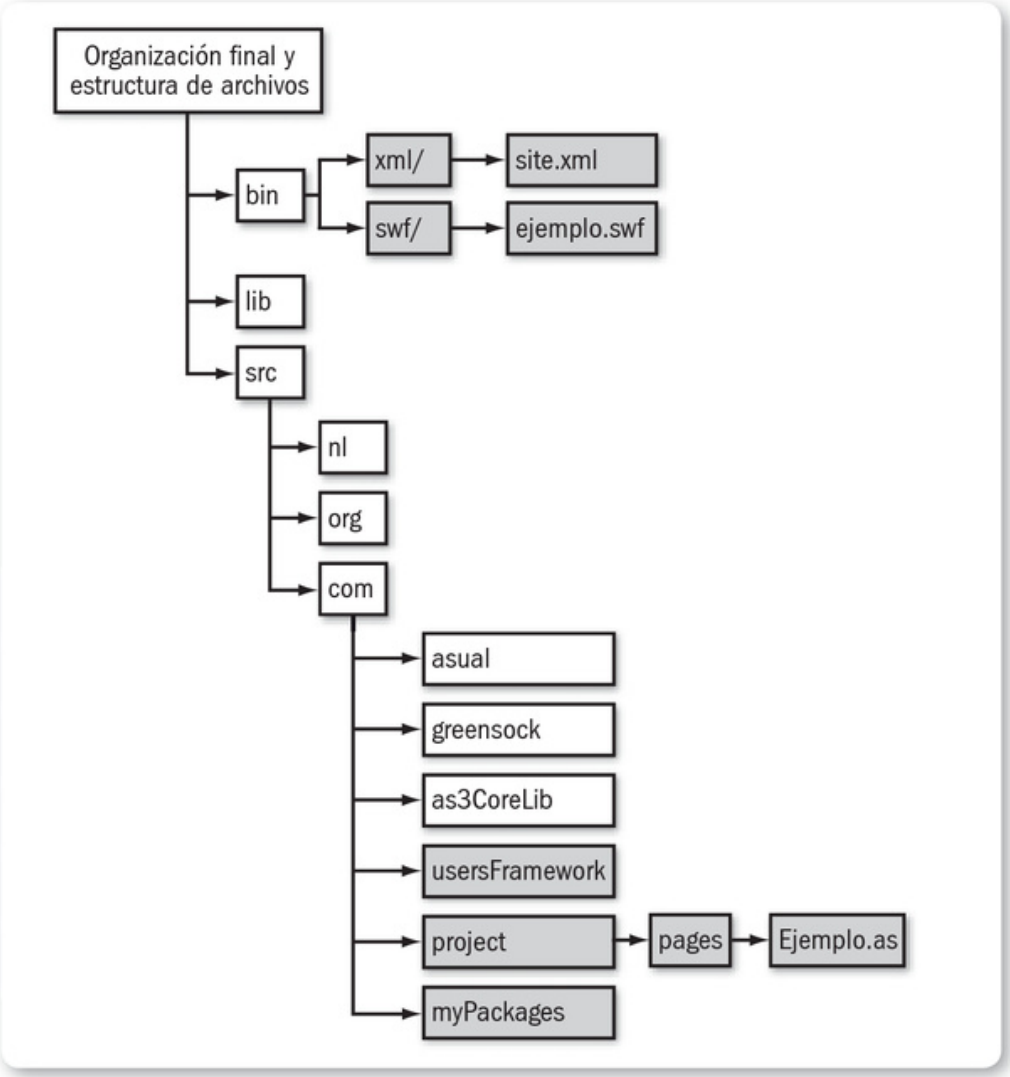



**Figura 9.** La evolución de nuestro espacio de trabajo, hasta el momento. A partir de este punto, inicia la programación de los contenidos.

## RESUMEN

Una vez concluida la organización de todo el contenido, tanto de los directorios como de los archivos que componen el proyecto, ya contamos con una estructura sólida y escalable. La lectura de este capítulo nos permite conocer por qué es tan importante mantener jerarquías dentro de nuestro trabajo y por qué cada elemento debe ir dentro de la carpeta que le corresponda. A su vez, comenzamos a ver cómo, a medida que crece un desarrollo, cada vez resulta más conveniente mantener una estructura estable.

# Actividades

## TEST DE AUTOEVALUACIÓN

1. ¿Qué clases incluiremos en el paquete **myPackage**?
2. ¿Qué clases incluiremos en el paquete **project**?
3. ¿Qué clases incluiremos en el paquete **usersFramework**?
4. ¿Cuál es la diferencia entre estos tres paquetes?
5. ¿Dónde radica la importancia de mantener este orden en la organización de los paquetes?
6. ¿Qué archivos irán en la carpeta **bin**, cuáles en la carpeta **src** y qué otros en la carpeta **lib**?
7. ¿De qué manera exportamos una película en otro directorio?
8. ¿De qué forma le indicamos a un archivo **.FLA** que busque su clase principal en otro directorio?
9. ¿Para qué utilizaremos el archivo **site.xml**? ¿Cuál es la diferencia entre los atributos del nodo **site** y los del nodo **page**?
10. ¿Dónde guardamos la clase principal del archivo **main.fla**, y dónde las clases principales de los archivos **.FLA** de las páginas del sitio por realizar?

## ACTIVIDADES PRÁCTICAS

1. Cree un proyecto en **FlashDevelop** desde cero.
2. Cree o edite el archivo **site.xml** que hemos generado en este capítulo, y asígnele cinco secciones distintas. Guarde el archivo en el lugar que corresponda.
3. Cree el archivo **.FLA** principal del proyecto, y asígnele como su clase principal el archivo **Main.as** creado por **FlashDevelop**.
4. Abra los archivos **.FLA** y modifique el directorio al cual deben exportar la película **.SWF**, para que esta se almacene en el directorio **bin/swf**.
5. Edite la configuración de los archivos **.FLA** para que busquen sus respectivas clases dentro del directorio **src**.

*04

# SEO en Flash: posicionamiento

El posicionamiento en buscadores es, tal vez, uno de los principales motivos por el cual se ha criticado el desarrollo de sitios en Flash durante mucho tiempo. En este capítulo veremos las herramientas con las que contamos para posicionar nuestro sitio, los mitos del SEO en Flash y la manera de obtener mejores resultados.

# Flash y SEO: mitos y verdades

**SEO** (sigla de *Search Engine Optimization*, o **posicionamiento en buscadores**) significa hacer uso de técnicas y herramientas para obtener una mejor visibilidad de nuestros sitios en los motores de búsqueda. Sabemos que hay formas pagas de lograrlo, y otras que son gratuitas y pasan por el buen uso y las buenas prácticas de las herramientas con las que contamos. Generalmente, se consideró el uso de Flash como una mala práctica cuando se habla de posicionamiento en la Web. A lo largo de este capítulo, veremos cuánto hay de cierto en esto, cuánto hay de mito, cuánta responsabilidad recae en nosotros como desarrolladores, cuánta en Flash y cuánta en HTML.

## ¿El contenido generado en Flash es indexable por los buscadores?

La mayoría de las personas (o todas) tienden a creer que los buscadores no son capaces de leer contenido dentro de una película SWF. Si bien ha sido de esta manera durante mucho tiempo, la realidad es que, actualmente, los principales motores de búsqueda son capaces de buscar dentro de nuestras películas el texto que hayamos incluido, igual que cualquier contenido HTML generado por Flash. Y a pesar de que no existe una única forma de hacer que todo el contenido dentro de una película se encuentre disponible para los buscadores, podemos hacer que cada elemento sea lo más visible posible empleando las técnicas adecuadas para cada caso particular.

### GENERALIDADES DE SEO

Es necesario tener en cuenta que existen muchos conceptos vinculados al mundo del SEO que se darán por sabidos o en los cuales no profundizaremos, porque no es ese el objetivo de este libro (SEO, SERPs, SEM, PageRank, popularity, relevance, etc.). Solamente nos focalizaremos en aquellas prácticas que hacen a las películas Flash más amigables para los buscadores.

## ¿El contenido de Flash es malo para los buscadores?

El contenido generado en Flash, al igual que cualquier otro, es la representación de la información que se quiere brindar al usuario. En algunas oportunidades, basta con HTML, y en otras, necesitamos hacer uso de Flash. Flash puede funcionar muy bien con criterios de SEO cuando se lo emplea de la manera adecuada.

# Crear contenido en Flash para SEO

A continuación, haremos uso de algunas técnicas que nos permiten lograr una mejor visibilidad de nuestras películas en los buscadores. Una de ellas recae sobre Flash, como en el caso de los metadatos, y otras requieren el uso de recursos externos: utilizaremos HTML y PHP para generar contenido alternativo.

## Flash Metadata

Desde la versión CS4 de Flash, contamos con la posibilidad de incluir metadatos en los archivos **.SWF**. Si bien esto resulta muy útil y práctico, lamentablemente, no es tan sencillo obtener buenos resultados: en la actualidad, los buscadores no logran acceder a esta información, pero es de esperar que lo hagan en el futuro. Es una buena práctica incluir este tipo de información dentro de cada película.

### IMPORTANCIA DE HTML PARA SEO

Independientemente de la capacidad de los motores de búsqueda de obtener información dentro de las películas .SWF, debemos saber que la mayor responsabilidad para lograr un buen posicionamiento recae en el HTML que embebe la película. Esto es lo que determina, principalmente, el posicionamiento. Veremos bastante sobre HTML a lo largo de este capítulo.

## INCLUIR METADATOS DENTRO DE UN ARCHIVO SWF

**01**

1. Cree un nuevo archivo de Flash e ingrese en el panel **Configuración de Publicación**. Hay dos maneras de hacerlo: por medio del atajo **CTRL+SHIFT+F12**, o bien desde el menú que encontramos en **Archivo/ Configuración de publicación....**

**02**

Ingrese a la solapa llamada **Flash**, y presione el botón **Información de Archivo...** dentro del apartado **Configuración de SWF**.

Una vez dentro del panel **Información del Archivo**, podemos agregar los siguientes metadatos a nuestra película:

### ETIQUETAS PARA FORMULARIOS

| METADATO | DESCRIPCIÓN |
|---|---|
| **Título del documento** | Nombre de nuestro archivo SWF. |
| **Autor** | Autor de dicho archivo |
| **Cargo del autor** | Cargo del programador que llevó a cabo el desarrollo. |
| **Descripción** | Breve descripción del archivo. |
| **Autor de la descripción** | Quién escribió la descripción del archivo. |
| **Palabras clave** | Palabras clave con las cuales queremos asociar el contenido de la película. |

| ▼ METADATO | ▼ DESCRIPCIÓN |
|---|---|
| **Estado de copyright** | Podemos indicar si el contenido es con copyright o de dominio público. |
| **Aviso de copyright** | Podemos agregarlo en caso de ser necesario. |
| **URL de información de copyright** | URL en caso de que nuestro desarrollo cuente con algún tipo de copyright. |

**Tabla 1.** Metadatos disponibles para agregar a una película **.SWF**.

Podemos darnos cuenta que se trata de un panel que cuenta con varias solapas más que pueden resultar de utilidad según el proyecto realizado: **IPTC**, **IPTC Extension**, **Datos de cámara**, **Datos de video**, **Datos de audio**, **SWF móvil**, **Categorías**, **Origen**, **DICOM**, **Historia** y **Avanzado**. Seguramente de aquí a unos años, implementar este tipo de contenido va a ser de gran ayuda en lo que a SEO respecta.

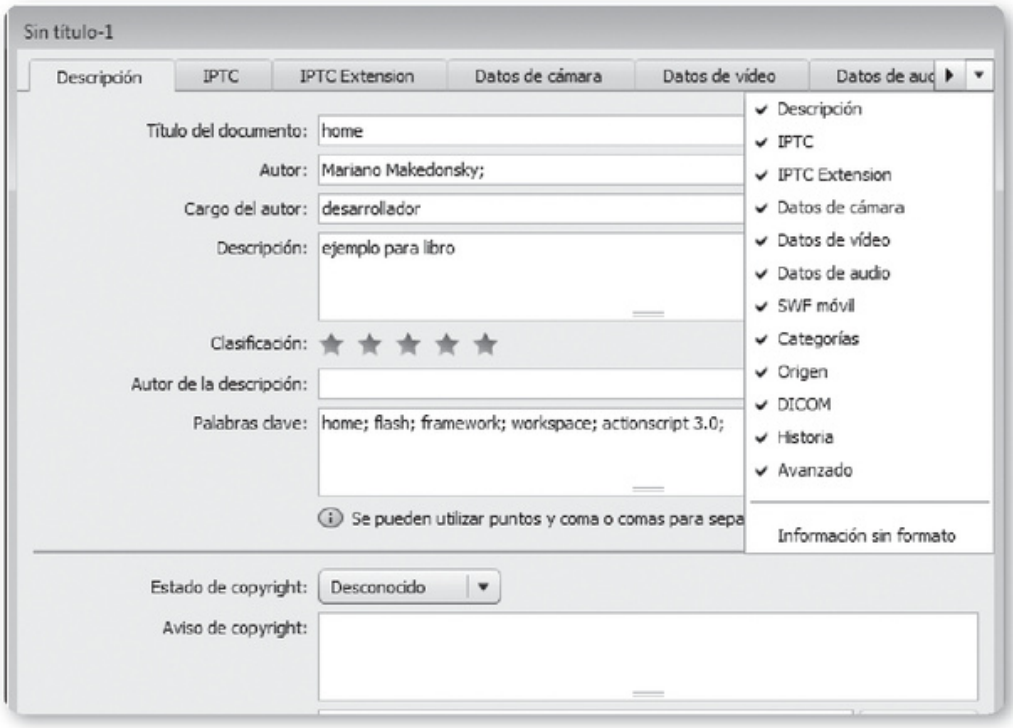



**Figura 1.** En esta imagen podemos ver el listado de opciones para metadatos con las que cuenta Flash CS5.

## ¿Cómo visualizar el contenido de los metadatos?

Desde el programa **Adobe Brigde**, podemos visualizar esta información seleccionando el archivo en cuestión. Es importante recordar que los buscadores actualmente no están accediendo a esta información, pero es de esperar que lo hagan en un futuro cercano.

A su vez, es necesario tener en cuenta que independientemente de los criterios de SEO, emplear los metadatos nos es de ayuda en vistas a mantener organizados nuestras películas Flash.

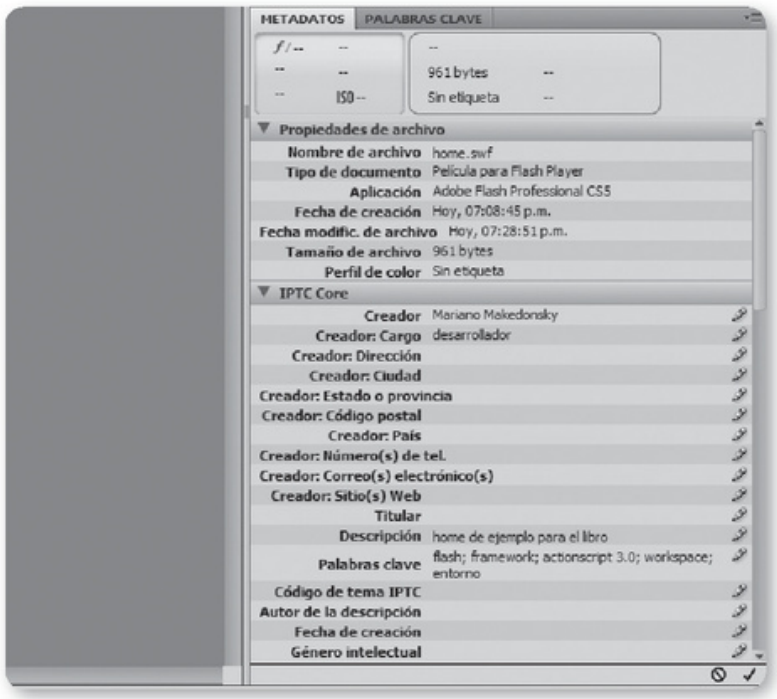



**Figura 2.** Una vez que seleccionamos el archivo que hemos creado dentro de **Adobe Bridge**, podemos ver sus metadatos desde el panel **METADATOS**.

### USO DE SWFOBJECT

SWFObject es la librería más utilizada al momento de embeber contenido dentro de un sitio. Durante el año 2010, se registraron más de 350.000 sitios que implementan esta metodología, incluyendo portales como **Windows.com**, **time.com**, **skype.com**, **discovert.com** y **youtube.com**. La gran ventaja de esta librería es que nos permite generar contenido alternativo para cuando no se encuentra instalado Flash Player.

# Contenido alternativo: SWFObject

Si creamos un nuevo proyecto con FlashDevelop, se genera una carpeta llamada **js** dentro del directorio **bin** de nuestra estructura. Allí encontraremos **swfaddress.js** y **swfobject.js**.

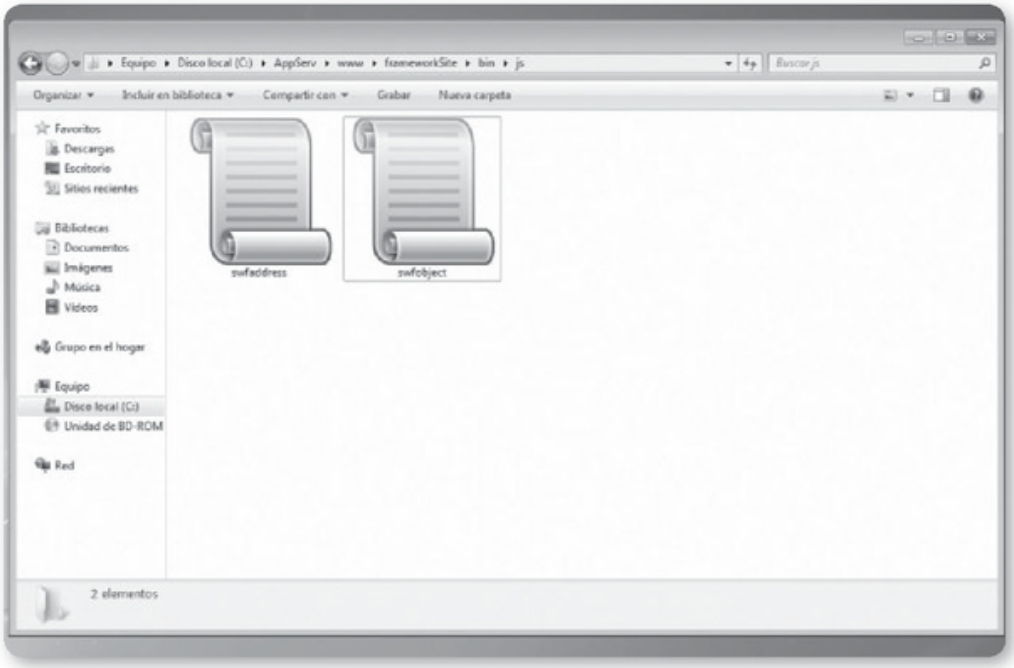



**Figura 3.** `FlashDevelop` se encarga de crear estos archivos por nosotros y los coloca dentro de la carpeta denominada `js`. A continuación, veremos cómo hacer uso de `SWFObject`.

## ¿Qué es SWFObject?

**SWFObject** es una librería de código abierto creada en JavaScript, que se utiliza para embeber contenido Flash dentro de una página. Permite detectar si hay una versión de Adobe Flash Player instalada en el navegador del usuario. En caso de no contar con el *plugin*, resulta sencillo mostrarle al visitante contenido alternativo o tomar una decisión, como puede ser redireccionarlo a otro sitio o informarle que necesita descargar Flash Player para poder continuar nuestro trabajo.


**Figura 4.** En caso de que no utilicemos **`FlashDevelop`** para gestionar un proyecto, podemos descargar la librería de SWFObject desde **`http://code.google.com/p/swfobject`**.

## SWFObject 2 HTML and JavaScript generator v1.2 (Adobe AIR 1 version)

Es necesario tener en cuenta que además de la librería, desde el sitio web de SWFObject es posibe acceder a la descarga de dos generadores de código: una versión creada en Adobe AIR, y otra en HTML. Trabajaremos con la versión en AIR para crear parte del código necesario para embeber el contenido dentro de nuestro sitio.

### GOOGLE ANALYTICS

UTengamos en cuenta que una de las maneras que podemos utilizar para medir los resultados que obtenemos de la implementación de técnicas para mejorar el posicionamiento de nuestro sitio es hacer un tracking de los contenidos en nuestros desarrollos. En próximos capítulos veremos de qué modo utilizar Google Analytics en un sitio generado íntegramente en Flash.


**Figura 5.** Desde **`http://code.google.com/p/swfobject`** también podemos descargar los generadores de código que SWFObject pone a nuestra disposición.

Debemos tener en cuenta que una vez que hemos realizado el proceso de descarga del programa y posteriormente lo instalamos, nos encontraremos con la siguiente interfaz.

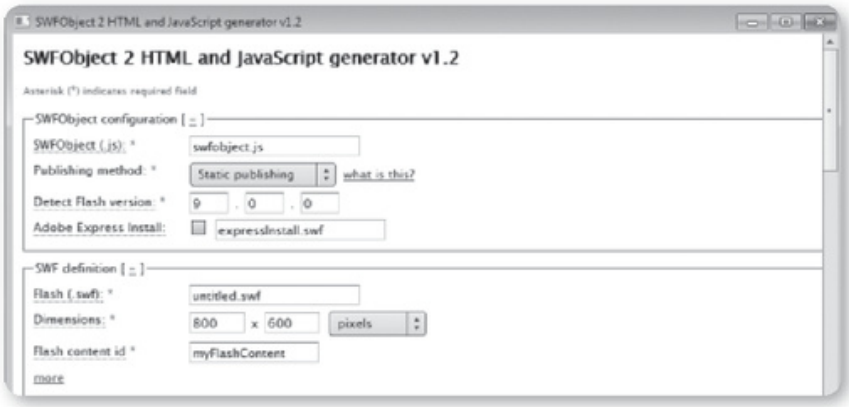



**Figura 6.** Veremos de qué manera sacar provecho de **`SWFObject 2 HTML and JavaScript generator v1.2`**.

Comencemos con la creación del código. Para realizar los siguientes pasos, debemos tener abierto el programa que acabamos de instalar. Veamos los campos disponibles en **SWFObject Configuration**.

Para continuar con esta tarea, en primer lugar, vamos a modificar la ruta de nuestro archivo JavaScript. Recordemos que se encuentra dentro de la carpeta denominada **js**, y nuestro archivo HTML estará por fuera de ella. Para que el HTML pueda acceder a él, debemos indicarle la ruta correcta del archivo **swfobject.js**: **js/swfobject.js**.

# SWFObject: publicación estática y publicación dinámica

**SWFObject** cuenta con dos modalidades para publicar contenido: una estática y otra dinámica, cada una de las cuales presenta ventajas y desventajas. Es importante entenderlas para elegir la metodología adecuada acorde a nuestras necesidades y poder emplear una opción u otra dependiendo de las características propias del desarrollo en el cual nos encontramos trabajando.

# Publicación estática

Por medio de la publicación estática, integramos Flash y el contenido alternativo empleando el marcado de lenguaje de hipertexto según los estándares **XHTML** y **JavaScript** para resolver cuestiones que, de por sí, solo puede resolver el lenguaje de marcado.

## Ventajas de la publicación estática

Las ventajas de este método de publicación son que se promueve el uso de estándares y, a su vez, el mecanismo mediante el cual embebemos el contenido Flash no depende de JavaScript.

## Desventajas de la publicación estática

No se resuelve el problema generado en los navegadores Internet Explorer 6+ y Opera 9+, en los cuales es necesario hacer clic sobre la película Flash para activarla (botón *click to activate*).

## Publicación dinámica

Por medio de la publicación dinámica, se inserta el contenido alternativo empleando el marcado de lenguaje de hipertexto según los estándares XHTML, y se integra Flash utilizando JavaScript.

### Ventajas de la publicación dinámica

Es más fácil de escribir, no contiene código redundante, y se evita el punto anteriormente mencionado como desventaja del método de publicación estática: no es necesario hacer clic sobre la aplicación para activarla en los navegadores Internet Explorer 6+ y Opera 9+.

### Desventajas de la publicación dinámica

El proceso mediante el cual se embebe el contenido Flash dentro del HTML recae sobre JavaScript. En este caso, por más que el usuario tenga Flash Player instalado, si está utilizando un navegador que no soporta JavaScript o bien este fue desactivado, no se podrá ver el contenido.

Nosotros trabajaremos con la publicación dinámica, pero la elección de la metodología queda a criterio de cada lector.

# Detección de la versión de FlashPlayer

A continuación, debemos indicar qué tipo de versión de Flash Player queremos que detecte SWFObject para considerarla válida y mostrar el contenido Flash, o bien mostrar el contenido alternativo. Como regla general, en la medida en que la versión por detectar sea más baja, mayores serán las posibilidades que tenemos de poder mostrar nuestra película **.SWF**. La desventaja que surge de esto es que, cuanto menor sea el número de versión de Flash Player para detectar, más bajas serán las posibilidades que nos brindan Flash y ActionScript para nuestro desarrollo. Lo que debemos hacer en estos casos es analizar en detalle nuestro proyecto en Flash, y saber si vale la pena consultar por la última versión de Flash Player, o bien podemos

PODEMOS RESOLVER UNA ANIMACIÓN SIMPLE CON FLASH PLAYER EN SU VERSIÓN 9.0

preguntar por una versión anterior, pero que, aun así, cumpla con los requisitos para mostrar nuestro contenido. En nuestro caso le diremos a SWFObject que detecte la versión **10.2.0**, que, al momento de escribir estas líneas, es la última estable disponible en Flash. Esto lo hacemos porque en nuestro proyecto utilizaremos recursos de ActionScript 3.0 que se encuentran disponibles solamente para esta versión.

Se trata de una nueva alternativa con la que contaremos a partir de la versión 11 de Flash Player a través de la cual podremos reproducir videos delegando este proceso al GPU de la computadora, optimizando considerablemente el rendimiento y la performance.

## Adobe Express Install

Finalmente, contamos con dos campos más: uno para indicar si queremos instalar la versión de Adobe Express Install, y otro para definir el nombre del div que llevará el contenido alternativo de nuestro sitio. Adobe Express Install fue una novedad que se introdujo en la versión 8 de Flash, a través de la cual se le da la posibilidad al usuario de actualizar su versión sin necesidad de abandonar el sitio. En estos campos dejaremos la información que viene por defecto.

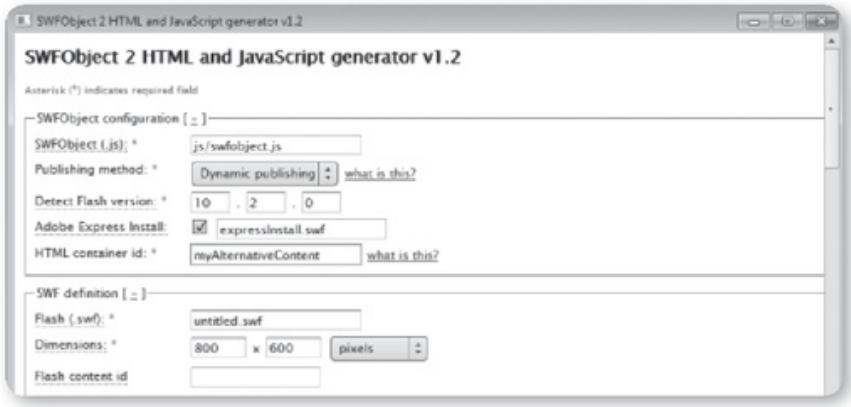



**Figura 7.** Toda la configuración que hemos establecido para nuestro sitio dentro de **`SWFObject Configuration`**.

Una vez concluidos estos pasos, modificaremos la información que se ubica dentro de **SWF Definition**, en relación a nuestra película. En primer lugar, debemos indicar el archivo por embeber (**swf/home.swf** en nuestro caso) y sus dimensiones. Utilizaremos el escenario al 100%, por lo que seleccionamos la opción **Percentaje** del menú desplegable. Si presionamos sobre el botón **more**, se presentará una serie de opciones adicionales con las que contamos a la hora de embeber el contenido. Por el momento, las dejaremos con sus valores predefinidos. Luego, lo haremos directamente desde el archivo HTML o PHP que utilicemos.

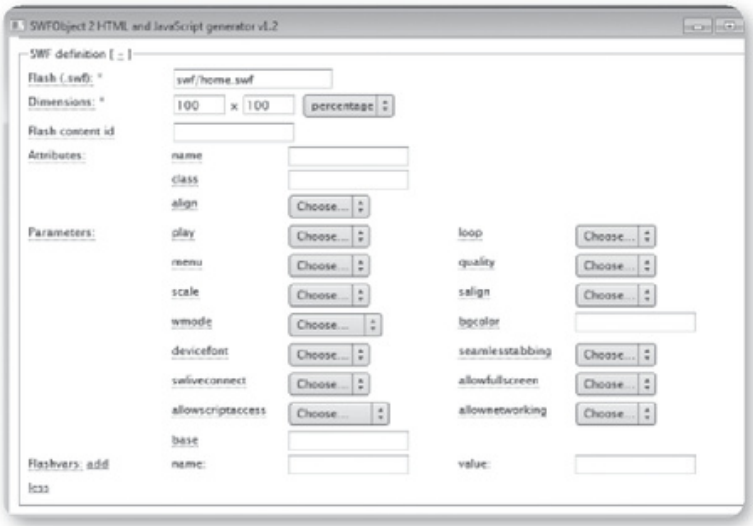



**Figura 8.** Opciones adicionales con las que cuenta **SWFObject** para embeber nuestro contenido SWF dentro de la estructura HTML.

SWF Generator cuenta con el apartado **HTML Definition**. Ahora no modificamos nada desde la interfaz en Adobe AIR. La posibilidad de crear contenido alternativo es la máxima ventaja de SWFObject. Este contenido en un sitio es importante para:

- Brindar información alternativa a quienes no cuentan con los *plugins* necesarios para visualizar Flash o lo ven desde un smartphone.

## GOOGLE WEBMASTER TOOLS

A través de este servicio, Google nos brinda informes detallados sobre la visibilidad con la que cuentan nuestros sitios en él. Google Webmaster Tools es una buena manera de mantener monitoreados los sitios y saber si nos está faltando algo para obtener un buen posicionamiento.

- Crear contenido SEO.
- Avisarle al usuario que no puede experimentar el sitio si no cuenta con Flash Player.

Una vez concluidos estos pasos, presionamos el botón **Generate**, y tendremos a nuestra disposición el código HTML que debemos incluir, el cual contiene todos los seteos que hemos indicado en estas páginas:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html;
            charset=iso-8859-1" />
        <script type="text/javascript" src="js/swfobject.js"></script>
        <script type="text/javascript">
            var flashvars = {};
            var params = {};
            var attributes = {};
            swfobject.embedSWF(„swf/home.swf", „myAlternativeContent",
                „100%", „100%", „10.2.0", „expressInstall.swf", flashvars,
                    params, attributes);
        </script>
    </head>
    <body>
        <div id="myAlternativeContent">
            <a href="http://www.adobe.com/go/getflashplayer">
                <img src="http://www.adobe.com/images/shared/download_buttons/
                    get_flash_player.gif" alt="Get Adobe Flash player" />
            </a>
        </div>
    </body>
</html>
```

Debemos crear un archivo HTML (que en las próximas páginas lo convertiremos en PHP) y pegarle el código que hemos generado.

# Consideraciones sobre el HTML de nuestro sitio

Como hemos dicho anteriormente, mucha de la responsabilidad respecto al éxito que obtenga nuestro sitio en lo que hace a la optimización de contenido para buscadores recae sobre el código HTML, y no, sobre Flash, como podriamos pensar.

## Título del sitio: tag <title></title>

El título del sitio es el aspecto más importante para tener en cuenta a la hora de pensar en el posicionamiento en buscadores. Esta información se edita desde el HTML. El título y las metaetiquetas del HTML son los únicos dos elementos de la lista de utilidades con la que contamos para SEO sobre las cuales tenemos un control absoluto. Por este motivo, independientemente de si el sitio está hecho en Flash o no, es imprescindible definir un buen nombre acorde al contenido que tendrá, y evitar el título **Untitled Page** que asignan los editores de HTML por defecto, lo cual es pésimo en vistas a obtener buenos resultados en cuanto al posicionamiento en buscadores.

> EL TÍTULO DEL SITIO ES EL ASPECTO MÁS IMPORTANTE PARA EL POSICIONAMIENTO EN BUSCADORES

## Metaetiquetas de HTML

Si bien en lo que fue el comienzo de este mundo llamado SEO las metaetiquetas fueron de vital importancia, hoy en día no tienen la misma relevancia que años atrás, y los motores priorizan otro tipo de información. Pero aun así, sigue siendo conveniente incluirlas. Si nos centramos exclusivamente en Google, allí se aclara que este motor de búsqueda procesa únicamente aquellas metaetiquetas que entiende, e ignora el resto. Es importante saber que, si bien son importantes, no son garantía de éxito. Por el contrario, los buscadores suelen utilizar metodologías bastante más complejas para medir la relevancia de nuestro sitio en la Web. No podemos abarcar este tema en su totalidad porque escapa a nuestro punto de interés, pero sí podemos recomendar

que los usuarios lean respecto a cómo los motores de Google procesan la información. En cuanto a los *metatags*, en el siguiente sitio podemos encontrar una lista detallada de cuáles interpreta Google y cuáles no, cuáles son comunes a todos los buscadores y cuáles son exclusivos para sus motores de búsqueda: **www.google.com/support/webmasters/bin/answer.py?answer=79812**.

A su vez, en la Web hay disponible una inmensa cantidad de herramientas para generar estos códigos por nosotros, para luego copiarlos y pegarlos donde corresponda.

Ejemplo de metaetiquetas por utilizar:

```
<head>
<title>AS3 Framework</title>
<meta name="description" content="sitio web de prueba para libro de users, en
    el cual analizamos las problemÃ¡ticas de Flash y sus mejores soluciones" />
<meta name="keywords" content="Flash, AS3, Framework, Espacio de Trabajo,
    usabilidad, accesibilidad, SEO, optimizaciÃ³n" />
<meta name="author" content="Mariano Makedonsky, Fabricio Mouzo"
<meta name="robots" content="index, nofollow" />
<meta http-equiv="content-language" content="es">
</head>
```

# Ocultar información a los navegadores: robots.txt

En algunas oportunidades, hay contenido de nuestro sitio que no queremos que sea indexado por los buscadores. Para esto empleamos el archivo **robots.txt**, que permite restringir el acceso de los robots de

## METAETIQUETA KEYBOARD

La metaetiqueta **keyboard** ha sido, durante muchísimos años, una de las más utilizadas, pero la realidad es que los motores de Google no la leen. Aun así, sigue siendo importante emplearla, ya que tiene valor para otros buscadores, como es el caso de Yahoo!, que la sigue tomando en cuenta.

motores de búsqueda al contenido de nuestro sitio. Estos robots, antes de acceder a analizar nuestro sitio, verifican la existencia del archivo **robots.txt**. En caso de que este exista, lo analizan y ubican dentro de él qué contenido no indexar. En caso de que nuestro sitio no cuente con ningún área que sea susceptible de ser indexada, no es necesario crear este archivo. Se puede obtener más información respecto al uso de este archivo en **www.robotstxt.org/orig.html**. Un ejemplo claro de contenido que podemos ocultarle al navegador podría ser el administrador del sitio, si es que se cuenta con uno.

# Dar de alta nuestro sitio en buscadores

Debemos tener presente que aunque se trata de una tarea sumamente sencilla, es sorprendente la cantidad de desarrolladores que no dan de alta los sitios en los buscadores.

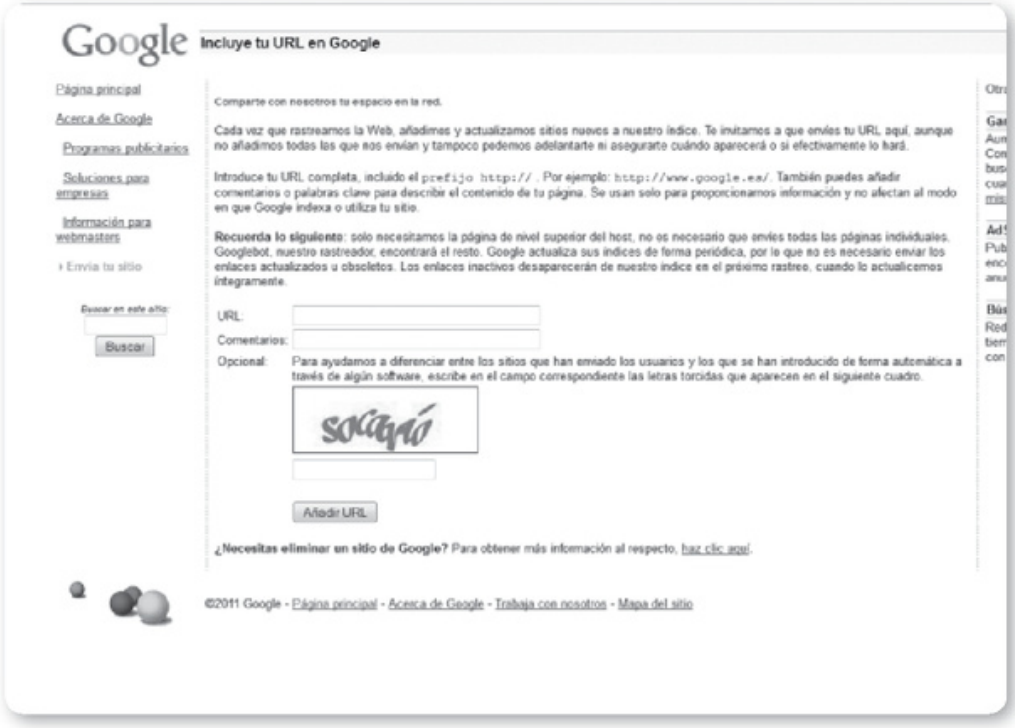



**Figura 9.** Desde el sitio que encontramos en **www.google.es/addurl** podemos dar de alta nuestro sitio web en Google.

Independientemente de las herramientas con las que contamos, todos los motores de búsqueda nos dan la opción de indicar manualmente la dirección de nuestro sitio y dar una breve descripción de él. Por esta razón, si bien ninguno nos garantiza que el sitio vaya a ser indexado, nunca está de más hacerlo.

# HTML y PHP: contenido alternativo para Flash

Anteriormente nombramos tres puntos que destacan la importancia de disponer de contenido alternativo para nuestro sitio. Debemos utilizarlo en aquellos casos en los que:

- Los usuarios no tienen Flash.
- Deseamos generar contenido para SEO.
- Queremos indicarle al usuario que hace falta tener Flash Player para ver el contenido.

Sea cual sea el motivo que tengamos, las tres opciones son válidas, y si bien tiempo atrás la realidad era distinta, el desarrollo moderno nos exige tomar en consideración las tres alternativas.

## Div para contenido alternativo

De esta forma, tengamos presente que SWFObject crea un **div** dentro del cual podemos asignar el contenido alternativo para nuestro sitio. Nosotros lo que haremos será personalizar este div, y dividirlo en dos; prestémosle atención a la siguiente estructura:

```
<div id="altContent">
<div id="navMenu">
        </div>
        <div id="noFlash">
        </div>
</div>
```

Dentro del div **altContent**, podemos darnos cuenta de que tenemos otros dos divs: uno destinado al menú del sitio (**navMenu**), y otro para informarle al usuario que no tiene Flash Player (**noFlash**).

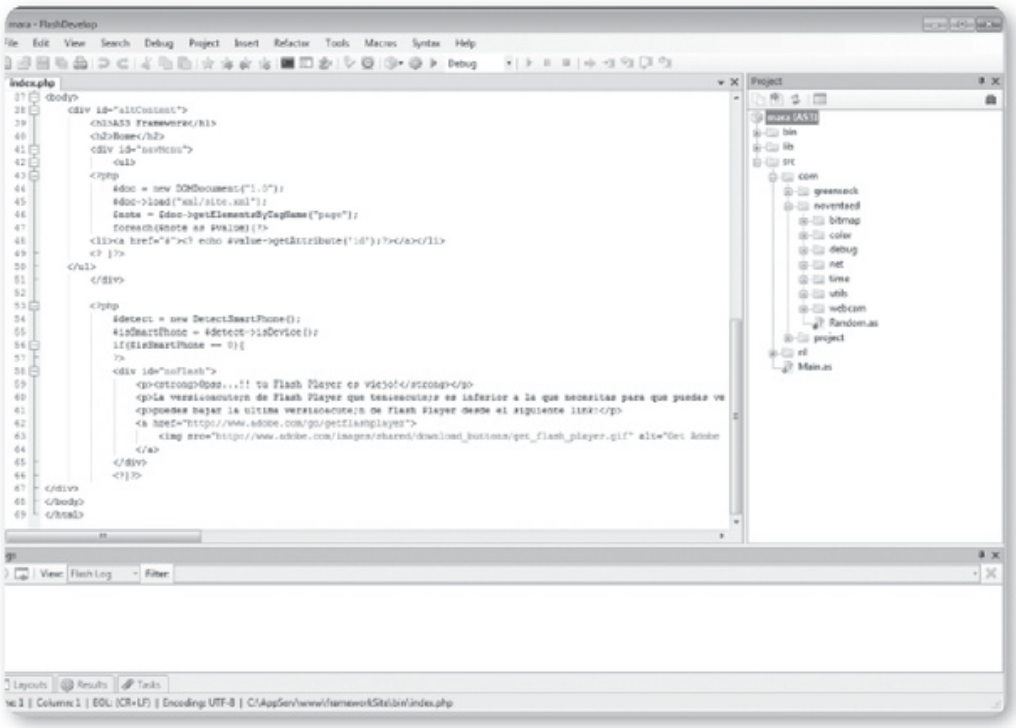


**Figura 10.** Vista del contenido alternativo de nuestro sitio. Luego veremos cómo generarlo y utilizar PHP para aprovechar su estructura.

# Menú alternativo en PHP

La importancia de contar con un menú alternativo para nuestro sitio es dual. Por un lado, nos da la posibilidad de brindarles a los usuarios que no tienen Flash una opción para navegar el sitio. Por otro, es sumamente útil en lo que respecta a SEO: pensemos que estaremos

## ROBOTS.TXT Y SITEMAP.XML

Si bien hemos visto que utilizamos el archivo **robots.txt** para indicarles a los buscadores qué contenido no indexar, en las próximas páginas veremos que contamos con una herramienta que nos sirve, justamente, para señalarles cuál es el que queremos indexar: **sitemap.xml**.

creando en HTML exactamente el mismo contenido que vamos a utilizar dentro de nuestras películas Flash (**site.xml**). Con este objetivo, debemos utilizar PHP a fin de parsear la estructura XML que hemos creado en el capítulo anterior, y generar un menú en HTML.

Para poder continuar, debemos cambiar la extensión del archivo que creamos anteriormente, de HTML a PHP. No ahondaremos en PHP, por lo que recomendamos que los lectores de esta obra cuenten con un *background* de conocimientos previos. Para que los archivos PHP funcionen correctamente, se deben ejecutar en un servidor local.

```
<div id="altContent">
    <h1>AS3 Framework</h1>
    <h2>Home</h2>
```

El tag denominado **<h1>** es el de mayor importancia dentro de la jerarquía de los elementos HTML, seguido por el tag llamado **<h2>**. En este caso, los utilizaremos para el título del sitio (**AS3 Framework**) y para el título de la sección actual (**home**).

PHP es un lenguaje sumamente potente, que, en verdad, resulta de gran utilidad como complemento de Flash para muchos tipos de desarrollos. Veamos cómo crear un menú alternativo en él:

```
<div id="navMenu">
    <ul>
        <?php
        $doc = new DOMDocument("1.0");
        $doc->load("xml/site.xml");
        $note = $doc->getElementsByTagName("page");
        foreach( $note as $value ) {?>
            <li><a href="#"><?echo $value->getAttribute('id');?></a></li>
        <? }?>
    </ul>
</div>
```

La clase **DOMDocument()** de PHP nos permite representar un documento entero HTML o XML. Una vez creada una instancia de la

clase, debemos indicarle el archivo por cargar. Lo que haremos será levantar el mismo archivo que vamos a utilizar dentro de Flash para generar nuestras páginas: **site.xml**.

```
$doc = new DOMDocument("1.0");
$doc->load("xml/site.xml");
```

Con estas dos líneas, tenemos almacenada en la variable **$doc** la estructura XML que contiene la información de nuestro sitio y de sus páginas. Para recrear el menú, debemos saber con qué páginas contará el sitio, y para esto, tenemos que acceder a los nodos de la estructura XML cuyos nombres sean **page**. Esto se lleva a cabo de la siguiente manera:

```
$note = $doc->getElementsByTagName("page");
```

Lo único que resta por hacer es un bucle **for**, y por cada elemento **page** que encontremos dentro de la estructura XML, generaremos un link en código HTML que apunte a su respectiva sección:

```
foreach( $note as $value ) {?>
        <li><a href="#"><?echo $value->getAttribute('id');?></a></li>
<? }?>
```

En lo que respecta a la optimización del contenido para buscadores, esto representa una inmensa ventaja. Cada vez que se ejecute el archivo **index.php**, se escribirá como código HTML una botonera con el menú que creamos dentro de nuestro archivo XML, que próximamente

## TAMAÑO DE LOS ARCHIVOS

Las buenas prácticas en lo que hace a usabilidad, accesibilidad y optimización de contenidos repercutirán de forma positiva en nuestro desarrollo. A su vez, se recomienda que el tamaño de los archivos HTML no supere los 150 KB, incluyendo el contenido que estos embeben.

utilizaremos dentro de Flash. A su vez, trabajar de este modo es sumamente ventajoso a la hora de optimizar tiempos y procesos de desarrollo: a partir de este momento, si queremos agregar una o diez secciones más a nuestra página, lo único que haremos será editar la estructura XML, y nuestro archivo PHP generará, sobre esa base, un menú en HTML, independientemente de la cantidad de elementos con los que cuente nuestro menú. El mismo material que utilizamos para el contenido del sitio Flash (veremos cómo cargarlo en el próximo capítulo) también lo estamos usando por medio de PHP para crear información en HTML para los buscadores. Interesante, ¿no?

**Figura 11.** Si ejecutamos el archivo PHP dentro de un servidor local o dentro de una web, veremos el menú de nuestro sitio.

Es importante tener en cuenta que no abarcaremos criterios de diseño, ni nos adentraremos en todo lo referente al maquetado y los estilos de CSS y HTML. Lo fundamental es saber que, con estas pequeñas líneas en PHP, ya tenemos a nuestra disposición un menú generado de forma dinámica, listo para que cualquier buscador lo pueda indexar.

## PHP Y FLASH

Además de ser un lenguaje de programación robusto, PHP es de carácter *open source* (de código abierto). A su vez, la sencillez de su sintaxis y las facilidades que brinda Flash a la hora de enviar y recibir datos convierten a PHP en la opción ideal para trabajar del lado del servidor en proyectos realizados en Flash.

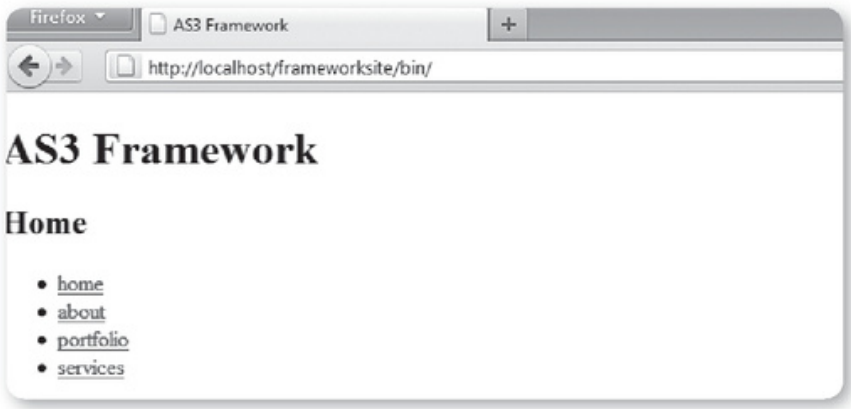



**Figura 12.** Cuando se ingrese desde un navegador que no soporte Flash o desde un smartphone, se mostrará el contenido alternativo.

# Flash en smartphones

Si bien **Android** dio el primer paso al incluir contenido SWF dentro de su navegador, hoy en día hay una importante cantidad de teléfonos **inteligentes** que no pueden reproducir contenido Flash. El uso de estos equipos creció de forma exponencial, y muchas veces necesitaremos disponer de contenido alternativo para los usuarios que accedan desde ellos y no puedan reproducir archivos SWF.

## Smartphones y computadoras sin Flash Player

Según los índices de penetración de Flash Player, el 98% de las computadoras cuenta con la versión 10 instalada. Años atrás, el hecho de detectar si el usuario tenía Flash o no resultaba útil para indicarle a ese 1% o 2% de los equipos que no lo tenían que era necesario para ver un sitio, o bien que hacía falta una actualización para visualizar el contenido correctamente (hablamos antes de **Adobe Express Install** y su importancia). Hoy en día, la realidad es muy distinta. Más importante que ese **1% o 2% de las computadoras que no cuentan con Flash**, son los **smartphones que no pueden correr este tipo de contenido**. Y en este caso, la cifra se

> EL 98% DE LAS COMPUTADORAS CUENTA CON ADOBE FLASH PLAYER EN SU VERSION 10

incrementa de manera considerable. Es por este motivo que resulta más útil brindarle contenido alternativo a un usuario que se conecta desde este tipo de dispositivo, que decirle que tiene que bajar una versión de Flash Player cuando, quizá, su smartphone ni siquiera acepte Flash, como en el caso del dispositivo **iPhone**.

Entonces, ¿en qué caso resulta práctico decirle al usuario que necesita bajar una nueva versión de Flash Player? Solamente cuando sepamos que está accediendo al contenido desde una computadora en la cual puede bajar una versión de Flash Player. ¿En qué caso resulta práctico dejar el contenido alternativo de nuestro sitio? Cuando detectamos que alguien está accediendo a él desde un *smartphone*, y probablemente su dispositivo no pueda correr ninguna versión de Flash Player.

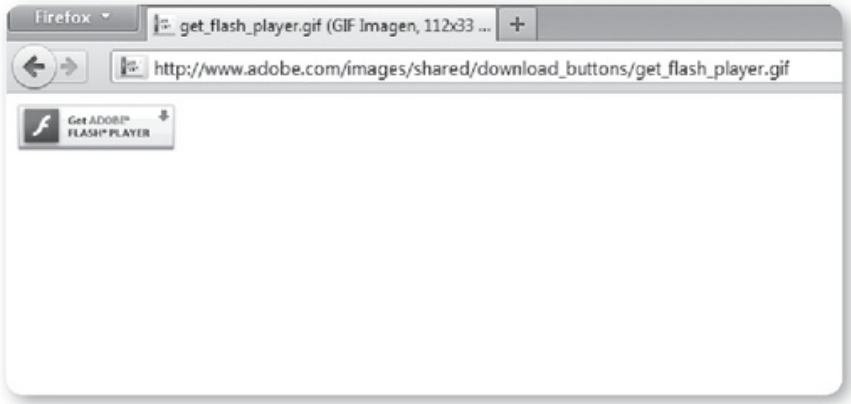



**Figura 13.** Imagen de Adobe utilizada generalmente para indicarle al usuario que debe bajar Flash Player para ver el contenido.

# PHP: clase DetectSmartPhone.php

Reiteramos que PHP no es el objetivo de este libro, pero es de esperar que los lectores estén familiarizados con su sintaxis y con la estructura de una clase. Analicemos su contenido:

```
<?php

/**
* ...
```

```
* @author #90ED.
* Mariano Makedonsky
*
* Clase para detectar si se está accediendo al sitio desde un dispositivo
*/

class DetectSmartPhone
{
    var $userAgent = "";

    var $true = 1;

    var $false = 0;

    var $devices = array('iPhone', 'Android', 'blackBerry', 'SymbianOS',
        'IEMobile', 'Maemo', 'webOS');

    /**
    * ...
    * DetectSmartPhone()
    * constructor de la clase. Asignamos la cadena $_SERVER['HTTP_USER_
        AGENT'] a la variable
    * $userAgent de esta clase y la pasamos a minuscula
    */

        function DetectSmartPhone(){
    $this->userAgent = strtolower($_SERVER['HTTP_USER_AGENT']);
 }

 /**
    * ...
    * isDevice()
    * @return true/false en función de si es un dispositivo móvil. No importa cual
    */

    function isDevice() {
        foreach ($this->devices as $device) {
```

```
            if (strpos($this->userAgent, $device) !== false) return $this->true;
        }
    return $this->false;
  }

  /**
    * ...
    * getUserAgent()
    * @return $userAgent en caso de que queramos acceder al userAgent
    */
    function getUserAgent(){
    return $this->userAgent;
  }
}
?>
```

Una vez declaradas las variables que se utilizarán dentro de la clase, accedemos al constructor de ella:

```
function DetectSmartPhone(){
        $this->userAgent = strtolower($_SERVER[`HTTP_USER_AGENT']);
}
```

Dentro del constructor, igualamos la variable **$userAgent** a la cadena **$_SERVER[‘HTTP_USER_AGENT’]** .

## Detectar iPhone, Android, BlackBerry, y más...

La cadena de texto *User Agent* nos brinda información sobre la aplicación que está accediendo a la página. Cada vez que se hace un pedido a ella, el *User Agent* se identifica comunicándole al servidor una cadena de texto, a la cual podemos acceder de manera sencilla a través de PHP. En nuestro código lo hemos hecho de la siguiente forma:

```
$this->userAgent = strtolower($_SERVER[`HTTP_USER_AGENT']);
```

Ahora bien, lo que nos queda por averiguar es si la cadena en cuestión contiene alguna palabra clave mediante la cual podamos identificar que el dispositivo que se está conectando es un *smartphone*:

```
var $devices = array('iPhone', 'Android', 'blackberry', 'SymbianOS', 'IEMobile',
'Maemo', 'webOS');
```

A continuación, enumeramos las palabras clave por las cuales preguntaremos:

- **iPhone**: la palabra clave identifica al dispositivo como un iPhone o un iPod (no así el iPad).
- **Android**: el sistema operativo del dispositivo es Android.
- **BlackBerry**: el dispositivo que se ha conectado al sitio es un BlackBerry.
- **IEMobile**: se utiliza para dispositivos móviles con Windows Mobile.
- **SymbianOS/Maemo**: keyboards para identificar sistemas operativos de Nokia.
- **webOS**: palabra clave para identificar el sistema operativo de una Palm.

> USAREMOS WEBOS PARA IDENTIFICAR EL SISTEMA OPERATIVO INSTALADO EN UNA PALM

La ventaja de organizar el código de este modo es que, en caso de que salga un nuevo dispositivo, simplemente bastará con agregarlo al array **$devices** de nuestra clase:

```
var $devices = array('iPhone', 'Android', 'blackBerry', 'SymbianOS', 'IEMobile',
'Maemo', 'webOS');
```

## BLACK HAT SEO

A diferencia de White Hat SEO, se conoce como **Black Hat SEO** a aquellas prácticas no éticas por las cuales se busca obtener un mejor posicionamiento. Este tipo de prácticas son peligrosas para nuestros sitios, considerando que pueden ser eliminados y no tenidos en cuenta por los motores de búsqueda.

De la siguiente manera, validamos si el *User Agent* contiene alguna de las palabras clave que hemos indicado:

```
function isDevice() {
    foreach ($this->devices as $device) {
        if (strpos($this->userAgent, $device) !== false) return $this->true;
        }
    return $this->false;
}
```

Si se encuentra una de estas palabras, esta función devolverá **0**; en caso contrario, devolverá **1**. Lo que nos resta por hacer es emplear esta clase desde el archivo **index.php** que hemos editado en este capítulo.

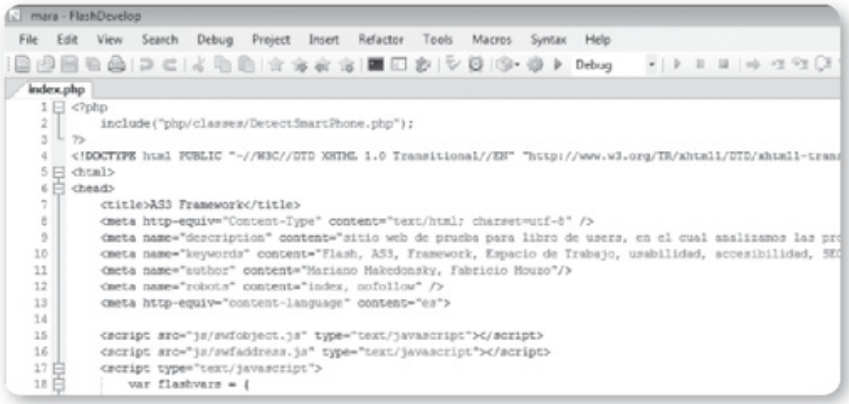



**Figura 14.** Una vez modificado el archivo, debemos incluir la clase que acabamos de crear.

Dentro de él incluimos la siguiente clase:

```
<?php
    include("php/classes/DetectSmartPhone.php");
?>
```

Anteriormente dijimos que es útil decirle al usuario que necesita descargar una versión de Flash Player o actualizarla si se ha conectado

desde una computadora. Es necesario tener en cuenta que esto podemos averiguarlo de la siguiente manera:

```
$detect = new DetectSmartPhone();
$isSmartPhone = $detect->isDevice();
if($isSmartPhone == 0){
```

En primer lugar, creamos una instancia de la clase **DetectSmartPhone()**. Luego creamos la variable **$isSmartPhone**, y la utilizamos para asignarle el valor que devuelve la función **isDevice()** de la clase **DetectSmartPhone()**. Recordemos que era **0** en caso de no ser un *smartphone*, y **1** en caso de no serlo.

Si es **0**, sabremos que el usuario no está accediendo a la Web desde un dispositivo móvil. Entonces, se trata de una computadora, y cobra sentido indicarle que debe descargar la última versión de Flash Player:

```
<?php
    $detect = new DetectSmartPhone();
    $isSmartPhone = $detect->isDevice();
    if($isSmartPhone == 0){

?>
    <div id="noFlash">
        <p><strong>Opss...!! tu Flash Player es viejo!</strong></p>
        <p>La versi&oacute;n de Flash Player que ten&eacute;s es inferior a la
            que necesitas para que puedas ver el contenido o bien, no tiene
                JavaScript habilitado en tu navegador</p>
        <p>puedes bajar la ultima versi&oacute;n de Flash Player desde el
            siguiente link:</p>
        <a href="http://www.adobe.com/go/getflashplayer">
            <img src="http://www.adobe.com/images/shared/download_buttons/
                get_flash_player.gif" alt="Get Adobe Flash player" />
        </a>
    </div>
<?}?>
```


**Figura 15.** Si la PC no cuenta con la versión de Flash Player, veremos contenido alternativo en HTML y podremos descargarlo.

Si el usuario está accediendo al sitio desde un smartphone, le aparecerá solamente el menú alternativo que creamos anteriormente con PHP. Qué hacer a partir de ahí ameritaría escribir un nuevo libro, ya que el desarrollo de sitios para smartphones es un mundo aparte. Nuestra recomendación es generar un sitio distinto para estos dos casos puntuales: uno para cuando el usuario accede desde una computadora, y otro para cuando lo hace desde un smartphone. Al detectar desde dónde está accediendo, es posible redireccionarlo al sitio que corresponda y de esta manera brindarle una mejor experiencia al usuario y adaptar el contenido al dispositivo que esté empleando.

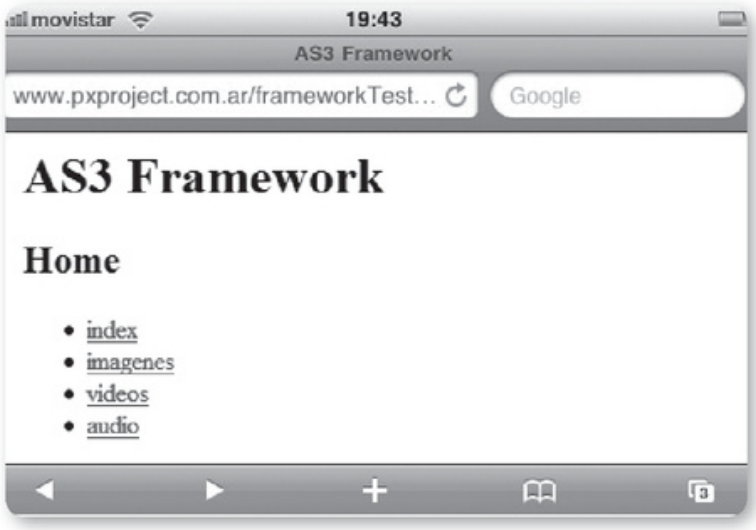



**Figura 16.** Si se accede desde un smartphone (en este caso, desde un iPhone), solamente se muestra el menú alternativo.

Por otro lado, si bien hemos dado el ejemplo del iPhone, que al momento de escribir estas líneas no permite visualizar películas SWF dentro de Safari, sí lo hace Android desde su versión 2.2. Haciendo unos pequeños cambios a la clase **DetectSmartPhone.php** creada en este capítulo, es sumamente sencillo diferenciar cuándo se está visitando el sitio desde un iPhone o desde un Android.

En caso de que se detecte un dispositivo con sistema operativo Android, podemos optar por mostrar el contenido del sitio, ya que soporta Flash. De todos modos, el desarrollo para teléfonos inteligentes es un mundo aparte: las resoluciones, las prestaciones, los recursos y los comportamientos son completamente distintos en un *smartphone*. Tengamos en cuenta que dedicaremos un capítulo de este libro a analizar esas cuestiones con mayor profundidad.

De todos modos, incentivamos a que aquellos desarrolladores que tengan un particular interés en el desarrollo para dispositivos móviles, se aboquen a un libro que trate el tema de forma específica.


**Figura 17.** Captura del sitio de **Grupo W** (**www.grupow.com**) desde un iPhone. La visualización es distinta que en una PC.

## WHITE HAT SEO

Se entiende por **White hat SEO** a las "buenas prácticas" que se llevan a cabo teniendo como finalidad lograr buenos resultados en lo que respecta a la optimización de contenidos para buscadores. Todas las técnicas descriptas en este capítulo y empleadas a lo largo del libro se ubican dentro de este grupo.

# Más contenido para los buscadores: sitemap.xml

El **sitemap.xml** es un archivo XML dentro del cual podemos incluir la lista de páginas que queremos que sean accesibles para los motores de búsqueda. Estos archivos son especialmente útiles en aquellos casos en los cuales se trabaja con contenido Flash. La estructura que presenta el archivo **sitemap.xml** es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url>
        <loc>http://www.ejemplo.com/</loc>
        <lastmod>2013-01-01</lastmod>
        <changefreq>monthly</changefreq>
        <priority>0.8</priority>
</url>
</urlset>
```

De todos los nodos de la estructura XML, el único obligatorio es **<loc></loc>**, mediante el cual indicamos la URL por indexar:

```
<loc>http://www.ejemplo.com/</loc>
```

El resto del contenido del archivo **sitemap.xml** es opcional. En el sitio **www.sitemaps.org/protocol.php** podemos encontrar qué valores indicar para **<lastmod>**, **<changefreq>** y **<priority>**.

## MÁS SEO EN EL LIBRO

Todo lo relativo a SEO y Flash abarca varias facetas más de las que podemos cubrir en este capítulo. Si bien durante estas páginas veremos los contenidos más importantes, en los próximos capítulos iremos enriqueciendo lo que se refiere a la optimización de contenido para navegadores.

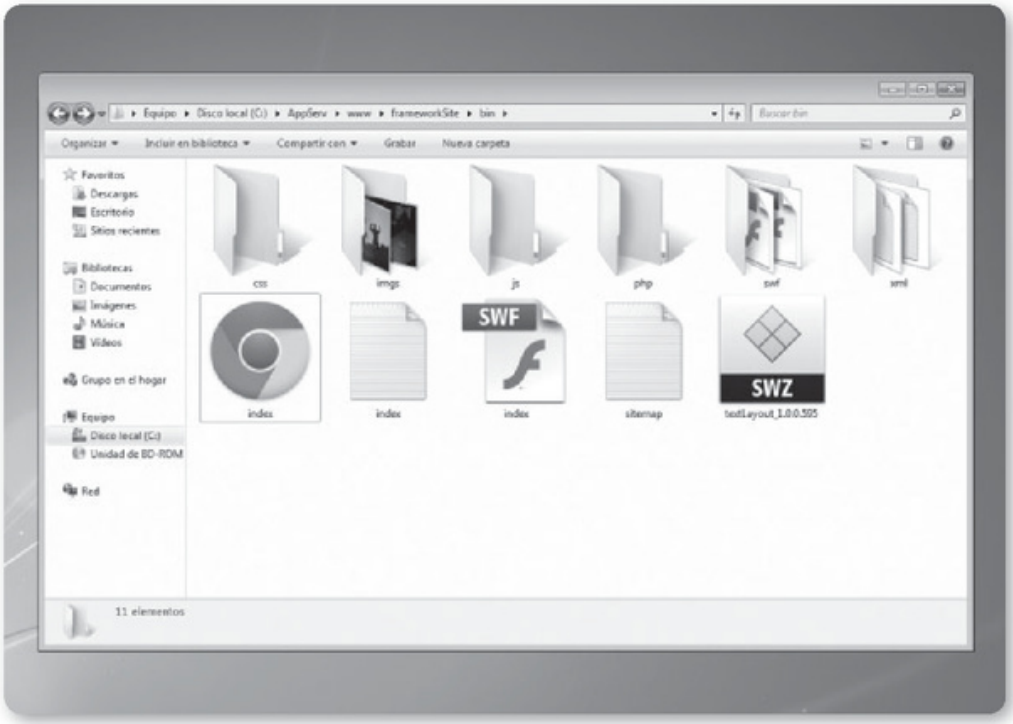



**Figura 18.** Debemos colocar el archivo **`sitemap.xml`** en el directorio principal de nuestro proyecto (donde está el archivo **`index.php`**).

Tengamos en cuenta que tenemos dos formas de crear el contenido que luego vamos a volcar dentro del archivo **sitemap.xml**.

## Distintas páginas HTML o PHP para cada SWF

Una de las formas es crear un archivo HTML o PHP por cada película SWF que tengamos. La base está armada en el archivo **index.php** con el que hemos trabajado en este capítulo. Dentro de él debemos modificar el archivo SWF que queremos embeber. Si nuestro sitio tuviera cuatro secciones y contáramos con **home.swf**, **servicios.swf**, **galeria.swf** y **contacto.swf**, deberíamos proceder a crear un listado de archivos como el que sigue: **home.html** (o **home.php**), **servicios.html** (o **servicios.php**), **galeria.html** (o **galeria.php**) y **contacto.html** (o **contacto.php**).

Es importante que tengamos en cuenta que cada uno de estos archivos HTML o PHP tendría que embeber a su respectivo archivo **.SWF** (**home.php** embebe el archivo **home.swf**, etcétera). El este caso, el archivo **sitemap.xml** quedaría compuesto de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url>
      <loc>http://www.ejemplo.com/</loc>
</url>
<url>
      <loc>http://www.ejemplo.com/home.php</loc>
</url>
<url>
      <loc>http://www.ejemplo.com/servicios.php</loc>
</url>
<url>
      <loc>http://www.ejemplo.com/galeria.php</loc>
</url>
<url>
      <loc>http://www.ejemplo.com/contacto.php</loc>
</url>
</urlset>
```

## Página PHP que valide variables GET

Otra alternativa con la que contamos es utilizar un único archivo PHP, que dependiendo de las variables GET que reciba, luego sepa qué película **.SWF** embeber. Teniendo en cuenta las mismas secciones que en el ejemplo que mostramos anteriormente, el **sitemap** quedaría compuesto como vemos en el siguiente trozo de código:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url>
      <loc>http://www.ejemplo.com/</loc>
</url>
<url>
      <loc>http://www.ejemplo.com/?seccion=home </loc>
</url>
<url>
```

```
        <loc>http://www.ejemplo.com/?seccion=servicios </loc>
</url>
<url>
        <loc>http://www.ejemplo.com/?seccion=galeria</loc>
</url>
<url>
        <loc>http://www.ejemplo.com/?seccion=contacto </loc>
</url>
</urlset>
```

Independientemente de todo lo que hemos visto en este capítulo, en vistas a mejorar la visibilidad de nuestros contenidos para los buscadores, es imprescindible informarse, formarse y hacer uso de aquellos recursos que son considerados buenas prácticas dentro del mundo de SEO. Muchos recursos forman parte del mundo Flash, y muchos del mundo HTML. Debemos dominar a ambos por igual si queremos obtener buenos resultados.

## RESUMEN

Si bien SEO en Flash es un tema muy amplio, en esta capítulo buscamos sentar las bases de lo que debemos considerar para que nuestras películas .SWF sean lo más visibles posible para los distintos buscadores. La mayor parte de los recursos con los que contamos recaen sobre HTML y PHP, pero como hemos visto, con unos sencillos pasos, podemos lograr que la parte más importante de nuestro desarrollo sea fácilmente indexable por los buscadores.

# Actividades

## TEST DE AUTOEVALUACIÓN

1. ¿Pueden los buscadores indexar contenido Flash?
2. ¿Es útil incluir **metadatos** en una película Flash? ¿Por qué?
3. ¿Para qué utilizamos **SWFObject**? ¿Para qué sirve la aplicación SWFObject Generator?
4. ¿Cuáles son los tres puntos en los que recae la importancia de utilizar contenido alternativo?
5. ¿Cuán importante es el título de un sitio para **SEO**? ¿Y las metaetiquetas?
6. ¿Cómo les indicamos a los motores de búsqueda que no queremos indexar una parte del sitio?
7. ¿De qué manera creamos contenido alternativo?
8. ¿Cómo detectamos si se está ingresando en el sitio desde un smartphone?
9. ¿Para qué sirve la cadena **User Agent**?
10. ¿Para qué sirve el archivo **sitemap.xml**?

## ACTIVIDADES PRÁCTICAS

1. Cree un nuevo proyecto teniendo en consideración lo que hemos planteado hasta aquí respecto a la optimización de contenidos.
2. Agregue a cada película sus respectivos metadatos.
3. Cree por cada película un archivo HTML o PHP dentro del cual se pueda generar contenido alternativo.
4. Cree el archivo **sitemap.xml** para el sitio.
5. Busque información respecto a cómo crear un archivo **robots.txt** y excluya el contenido de una página para que no sea indexada por los motores de búsqueda.

*05 –

# OOP y Design Patterns: estructura interna

A la hora de hablar sobre optimización de recursos y reutilización, es imposible obviar los beneficios que nos brindan la Programación Orientada a Objetos y los patrones de diseño. En este capítulo veremos de qué manera estructurar el código de nuestro proyecto por dentro.

# Introducción a la temática OOP y patrones de diseño

Cuando los proyectos cobran relevancia, es recomendable que nos aboquemos a la metodología de desarrollo que ofrecen la Programación Orientada a Objetos y los patrones de diseño. En este libro no explicaremos los conceptos básicos sobre OOP ni veremos su parte teórica: es de esperar que el lector cuente con esa base. Pero sí haremos uso, de aquí en adelante, del paradigma de programación OOP (*Object-Oriented Programming*). Ahora bien, además de la Programación Orientada a Objetos, contamos con una herramienta importantísima: el diseño de patrones (o *design patterns*). Los patrones de diseño se aplican, en especial, al desarrollo de interfaces, y básicamente son la solución a un problema de diseño, al aplicar la misma solución que ya se haya aplicado a casos anteriores de índole similar. Otra de las características con las que debe contar un patrón de diseño es ser reutilizable y aplicable en otras oportunidades. Existen muchísimos patrones de diseño. En estas páginas explicaremos solo aquellos que vamos a utilizar a lo largo de nuestro desarrollo: **MVC** y **Singleton**.

## Inicializar el framework: Main.as y FrameworkMain.as

En el **Capítulo 4** mencionamos la clase **Main.as**, la cual es cargada por la película principal de nuestra aplicación. En general, esta clase se extiende a la clase **Sprite**. En nuestro caso, haremos un pequeño paso intermedio: la extenderemos a la clase **FrameworkMain.as**, y luego sí, a

### VARIABLES, GETTERS Y SETTERS

Debemos saber que al utilizar métodos descriptores de acceso, tanto el **getter** como el **setter** para obtener un dato pueden llevar el mismo nombre. Por este motivo, se acostumbra a diferenciar el nombre de la variable de sus descriptores por medio de un guión bajo o de un signo pesos. Por ejemplo, se utiliza la variable **_frameRate** para el *getter* y **frameRate** para el *setter*.

**Sprite**. Esto nos facilitará tener el código lo más segmentado posible, y en vez de tener que inicializar todas las clases del *framework* manualmente desde la clase **Main.as**, delegamos este proceso a **FrameworkMain.as**:

```
public class Main extends FrameworkMain
```

Colocaremos la clase **FrameworkMain.as** dentro del paquete **core** (núcleo) de nuestro *framework* (**usersFramework**).

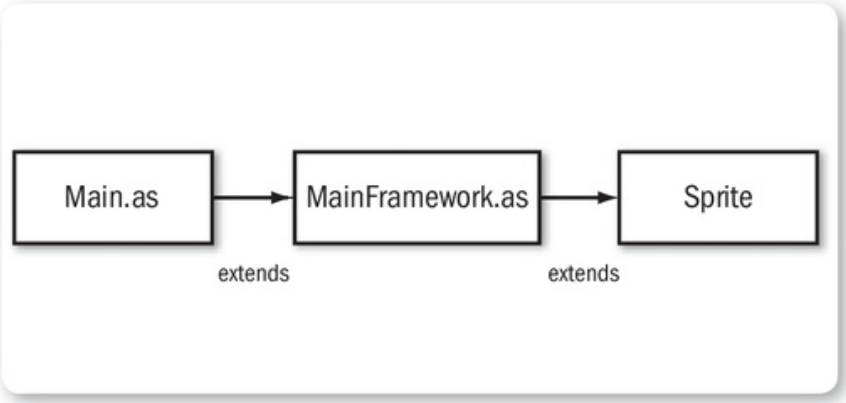



**Figura 1.** La clase principal de nuestra película (**`Main`**) extiende a **`FrameworkMain`**, y esta inicia la aplicación.

**FrameworkMain.as** inicializa todo el proceso que conforma el espacio de trabajo. En primer lugar, asignamos un evento para saber cuándo tenemos en nuestro código una referencia al escenario:

```
public function FrameworkMain():void {
    super();
    addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
}
```

Flash tiene una falencia en este sentido: no siempre que se ejecuta la función en respuesta al evento asignado se cuenta con el escenario. En este caso, removemos el evento anteriormente asignado (**removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage)**)

y preguntamos por las dimensiones del escenario: si su largo y ancho continúan siendo 0, asignamos un evento **ENTER_FRAME**; de lo contrario, llamamos a la función **startFramework()** y seguimos esperando porque se produzca un cambio en los valores del ancho y largo de nuestra película. Cuando esto suceda, ejecutamos la función startFramework(), tal como vemos en el siguiente código:

```
protected function onAddedToStage(event:Event):void
{
removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
(stage.stageWidth == 0 || stage.stageHeight == 0) ? addEventListener(Event.
    ENTER_FRAME, onWaitingForStage, false, 0, true) : startFramework();
}
```

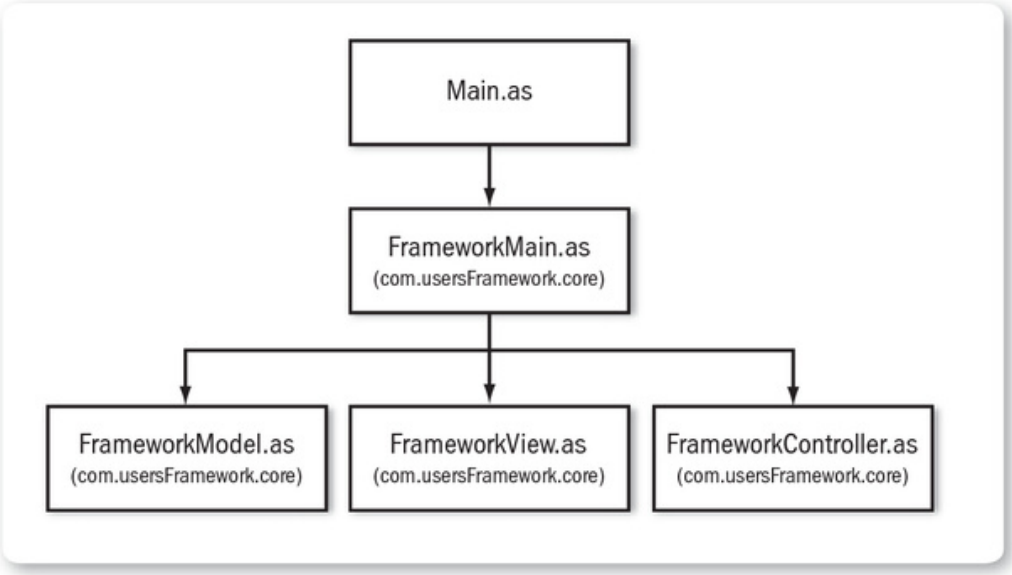



**Figura 2.** Desde **`FrameworkMain`**, iniciamos las tres clases que van a controlar el entorno de desarrollo: **`FrameworkModel`**, **`FrameworkView`** y **`FrameworkController`**.

El evento **ENTER_FRAME** continúa validando las dimensiones del escenario. Si siguen siendo 0, se sigue ejecutando hasta que estas cambien. La función que en verdad nos interesa es **startFramework()**, que se encargará de dar inicio al modelo de nuestro sitio:

```
frameworkModel = FrameworkModel.getInstance();
frameworkModel.addEventListener(Event.COMPLETE, onXmlLoadComplete,
    false, 0, true);
frameworkModel.loadXML(siteXML);
```

Para comprender qué es el modelo, en primer lugar tenemos que hacer una introducción al patrón MVC.

# Patrón MVC

MVC es la sigla de las palabras *Model*, *View*, *Controller*, o bien de Modelo, Vista, Controlador. Se trata de un patrón de diseño mediante el cual se separan los **datos** de la aplicación, la **interfaz** de usuario y el **control** de esta en tres partes distintas.

## Modelo

El modelo se encarga de contener la información de nuestra aplicación y, a su vez, maneja su estado.

> EL MODELO MANEJA EL ESTADO DE NUESTRA APLICACIÓN Y LA CONTIENE

## Vista

La vista es la interfaz de usuario, la parte visual de nuestro diseño. Cuando un usuario interactúa con nuestras aplicaciones, ya sea en un sitio web, en una aplicación móvil o en la plataforma que sea, lo hace a través de la vista de la aplicación.

## Controlador

Debemos tener en cuenta que el controlador es la parte del patrón mediante la cual se maneja el *input* correspondiente al usuario, y a través del cual podemos acceder a cambiar el estado de la aplicación correspondiente. A diferencia de la vista, el controlador reacciona sobre la base de las acciones que realice el usuario en la vista.

La mejor manera de ver qué responsabilidad le corresponde a cada elemento de este patrón es a través de un ejemplo. Pensemos en un reproductor de audio. En primer lugar, necesitamos que la información de las pistas ingrese en nuestra aplicación. Suponiendo que lo hagamos mediante una estructura XML, **el modelo se ocuparía de cargarla, parsearla y hacer que dicha información fuera accesible para nuestra aplicación**. A su vez, necesitaríamos de una vista, que sería la parte visual: los botones para pasar de tema, los botones para subir y bajar el volumen, los campos de texto para mostrar la información de cada pista, etcétera. Como dijimos anteriormente, el usuario interactúa con estos elementos. En caso de que se presione un botón, entra en juego el controlador de la aplicación, que, como hemos dicho, controla el *input* del usuario: entonces, lo que hace es determinar qué hacer con la acción que se llevó a cabo. Si el usuario subió el volumen (acción ejecutada desde la vista), se le informa al controlador que ocurrió dicha acción, este se ocupa de llevar a cabo el proceso que corresponda (en este caso, subir o bajar el volumen) y, una vez realizada la acción, le informa al modelo que se modificó el volumen. Recordemos que la otra acción que lleva a cabo el modelo, además de contener la información, es manejar el estado. Una vez que la información se modificó, se le avisa a la vista.

# MVC en conjunto

Si bien el modelo, la vista y el controlador tienen tareas bien distintas y específicas, trabajan en conjunto, y es necesario que las tres partes estén interconectadas. Precisamos tres clases separadas (una para el modelo, otra para la vista, y otra para el controlador), donde:

- El modelo requiere una referencia de la vista.
- La vista necesita referencias tanto del modelo como del controlador.
- El controlador necesita una referencia del modelo.

## SHORTCUT DE FLASHDEVELOP

Al crear un *listener* en FlashDevelop, si nos posicionamos sobre la función que este ejecuta y presionamos **CTRL+SHIFT+1**, se desplegará un menú por medio del cual podremos crear automáticamente el código correspondiente a dicho manejador, con la opción de que este sea de carácter público o privado.

```
model = new Model(view);
Controller = new Controller(model);
view = new View(model, view);
```

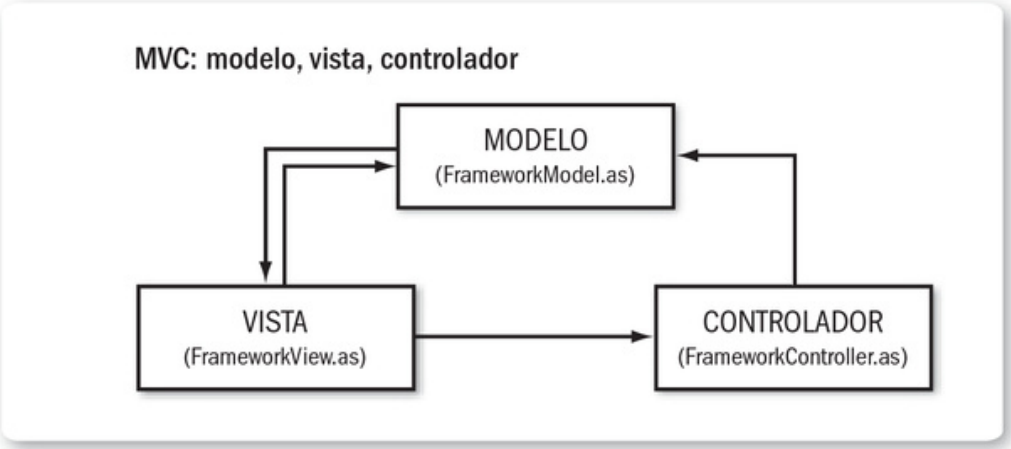



**Figura 3.** En esta imagen podemos ver el modelo de comunicación dentro del modelo MVC. Las partes del patrón tienen funciones específicas, pero necesitan estar comunicadas.

## Algunas consideraciones sobre el uso de patrones de diseño

El principal objetivo de un patrón de diseño es permitirnos crear código que podamos reutilizar en otros proyectos y nos deje realizar cambios de la manera más práctica y rápida. Esto no quiere decir que necesariamente nuestra aplicación vaya a tener un mejor rendimiento ni que vaya a presentar mejoras en su funcionamiento respecto a otras modalidades de desarrollo. De más está decir que el uso de *patterns* entra en juego cuando nuestras aplicaciones y desarrollos lo requieren. En muchos casos, no es preciso hacer uso de Programación Orientada a Objetos ni de patrones para resolver problemas, y esto depende, en gran medida, de la complejidad de nuestro trabajo. Pensemos en una galería de imágenes: una galería sencilla se puede resolver, incluso, sin líneas de código, de manera práctica, desde la línea de tiempo. No es necesario meterse con la Programación Orientada a Objetos ni con patrones para lograrlo. Ahora bien, en el supuesto caso de que un cliente nos pida una galería de imágenes donde la

información sea dinámica, haya varios álbumes, distintas maneras de visualización (tanto de los álbumes como de las fotos) e información específica sobre cada imagen, ya no nos alcanza con la línea de tiempo de Flash: de esta manera debemos tener en cuenta que necesitamos implementar una solución más inteligente y lograr que el código que utilicemos pueda emplearse en otras ocasiones para otros trabajos de índole similar: siguiendo estas indicaciones, en caso de que otro cliente precise alguna galería con características similares, ya contaremos con una base adecuada para ella, y tal vez con algunos pocos cambios, tengamos nuestro trabajo resuelto en forma completa.

# MVC para nuestro framework

Repasemos: el modelo cuenta con la información y modifica el estado. En nuestro caso, la información se encuentra dentro del XML que hemos creado en el **Capítulo 3** de este libro. La clase **FrameworkModel.as** se ocupará de cargar esta estructura, almacenar su información y hacerla accesible al resto de las clases del framework. La vista en nuestro desarrollo será cada una de las páginas que compongan el futuro sitio. Lo que hará la vista será contar con dos contenedores: uno para cargar cada una de las páginas, y otro para el *preloader*. Esto lo hará la clase **FrameworkView.as**. Por último, el controlador del sitio recibirá el *input* proveniente tanto de la botonera como del menú contextual, realizará los cambios necesarios y le avisará al modelo que llevó a cabo dicha acción, para que, luego, este modifique la vista. Esto lo hará la clase **FrameworkController.as**. Para manejar nuestro *framework*, necesitamos traer la información a la aplicación. Por esto mismo es que comenzaremos viendo la clase **FrameworkModel.as**:

```
private function startFramework():void {
    frameworkModel = FrameworkModel.getInstance();
    frameworkModel.addEventListener(Event.COMPLETE, onXmlLoadComplete,
        false,
    0, true);
    frameworkModel.loadXML(siteXML);
}
```

En primer lugar, recordemos que es vital prestar atención a la manera en la que inicializamos nuestra clase: **no lo hacemos mediante el constructor New, sino por medio del método getInstance()**, que contiene **FrameworkModel.as**:

```
frameworkModel = FrameworkModel.getInstance();
```

# Patrón Singleton

Utilizamos el patrón Singleton cuando necesitamos una única (y solo una) instancia de una clase en la aplicación. En nuestro MVC, sabemos que el modelo (**FrameworkModel.as**) almacena toda la información relativa al sitio. No tiene sentido contar con más de una instancia para esta clase: por el contrario, necesitamos que esta información sea la misma siempre, independientemente del lugar desde el cual accedamos, ya sea desde otras clases que componen el *framework*, o bien desde las páginas que componen un sitio. Pensémoslo de manera práctica: imaginemos un juego multiusuario en el cual hay una cuenta regresiva para que el juego termine. Independientemente de que cada usuario esté visualizando el contenido en distintas pantallas, es imprescindible que a ambos se les muestre la misma información respecto a la cuenta regresiva. La solución en este caso sería crear una única clase con una única instancia y un único punto de acceso, y que se pudiera ingresar en ella desde cualquier lugar. En nuestro *framework* sucede algo similar: la información que utilizamos debe ser en todos lados la misma, tanto si accedemos a ella desde otras clases del *framework* o si queremos leer o modificar dicha información desde cualquiera de las páginas que componen el sitio. Imaginemos que necesitamos el título del sitio en cada una de sus secciones. No tendría sentido crear distintas instancias para esto: solo creamos una clase con un punto de acceso, y hacemos que todas las páginas accedan a dicha instancia.

Hay dos características específicas que definen al patrón **Singleton**:

- Debe haber una única instancia de la clase, la cual puede ser instanciada en cualquier momento.
- La clase debe tener un único punto de acceso.

# Modelo: FrameworkModel.as

Anteriormente, vimos que la forma de inicializar la clase es llamando al método **getInstance()** de esta:

```
frameworkModel = FrameworkModel.getInstance();
```

Veamos el código que se encuentra dentro de dicha instancia:

```
private static var _instance:FrameworkModel;

/**
* ...
* getInstance():FrameworkModel
* @return instancia del modelo
*/

public static function getInstance():FrameworkModel
{
    if (_instance == null) return _instance = new FrameworkModel();
    return _instance;
}
```

En caso de que la variable **_instance** sea **null**, se nos devuelve una nueva instancia de la clase **FrameworkModel()**:

```
if (_instance == null) return _instance = new FrameworkModel();
```

## CONTENIDO Y DESARROLLO DE CLASES

Debido a la magnitud del espacio de trabajo por crear, es imposible abarcar la totalidad de las clases implicadas en este desarrollo. Solamente nos centraremos en lo más importante del *framework*. Por este motivo, es imprescindible contar con los archivos para poder seguir este capítulo.

Si **_instance** es distinto de **null**, quiere decir que ya existe una instancia de la clase. En este caso, devolvemos la variable **_instance**. A esto nos referíamos anteriormente al decir que un Singleton debe tener un único punto de acceso y que puede ser instanciado en cualquier momento. ¿De qué nos sirve esto? Al manejar una única instancia, podemos definir y obtener información sobre ella, sin importar la sección donde nos encontremos.

```
90      * constructor
91      */
92
93
94      public function FrameworkModel():void { }
95
96      /**
97      * ...
98      * getInstance():FrameworkModel
99      * @return instancia del modelo
100     */
101
102     public static function getInstance():FrameworkModel
103     {
104         if (_instance == null) return _instance = new FrameworkModel();
105         return _instance;
106     }
107
108     /**
109     * ...
110     * loadXML(path:String)
111     * @param path : la URL del XML a cargar.
112     * función que inicia la carga de la estructura XML que contiene los assets tanto
113     * del sitio como de las páginas que contendrá.
```

**Figura 4.** Punto de acceso de una clase Singleton. Usamos una función que nos devuelva una instancia de ella.

Si volvemos a la clase **FrameworkMain.as**, además de obtener la instancia de esta, asignamos un listener y llamamos al método **loadXML()**:

```
frameworkModel.addEventListener(Event.COMPLETE, onXmlLoadComplete,
    false, 0, true);
frameworkModel.loadXML(siteXML);
```

El listener nos informará cuando se completó la carga de la estructura XML (de esta manera, podremos terminar de conformar la tríada MVC), y el método **loadXML()** se ocupa de cargar la estructura XML con la información para nuestro sitio:

```
public function loadXML(path:String):void {
(path == null) ? this.xmlPath = "xml/site.xml" : this.xmlPath = path;
```

La función recibe como parámetro la cadena de texto con la ruta del XML. Si esta es igual a null, la igualamos a "xml/site.xml", que es donde se encuentra el archivo.

# Carga de XML: clase XMLDocumentLoader.as

Insistimos en reiteradas ocasiones respecto a la importancia de saber cuál es el alcance de una clase. Hasta el momento, todas las clases que venimos utilizando son útiles únicamente para nuestro *framework*. Ahora bien, una clase que se ocupa de cargar una estructura XML puede usarse en cualquier proyecto: su alcance es inmenso. Por este motivo, la clase no se encuentra dentro del paquete **usersFramework**, sino dentro de **MyPackages**. Ante cualquier tipo de proyecto, sabemos que nos podemos llevar el paquete **MyPackages** y tener dentro de él una serie de utilidades, como es el caso de la clase **XMLDocumentLoader.as**. La importancia de este libro radica en ir un poco más allá de lo que hace una clase que carga estructuras XML, pero este es un buen ejemplo para entender la **abstracción** dentro de la Programación Orientada a Objetos. No explicaremos el funcionamiento interno de la **XMLDocumentLoader.as**, pero justamente de eso se trata la abstracción: es la posibilidad con la que contamos de utilizar algo por medio de sus parámetros, sin que haga falta conocer los detalles de su funcionamiento. Quien quiera entender la manera en la cual funciona la clase solamente debe abrirla y analizarla. A nosotros nos importa cómo, con cuatro líneas de código, obtenemos una estructura XML:

```
xmlLoader = new XMLDocumentLoader(NoCache.create(path, Constants.
    IS_ONLINE));
xmlLoader.addEventListener(Event.COMPLETE, onXmlComplete, false, 0, true);
xmlLoader.addEventListener(IOErrorEvent.IO_ERROR, onIOError, false, 0, true);
xmlLoader.loadDocument();
}
```

El constructor de la clase **XMLDocumentLoader.as** recibe dos parámetros: por un lado, una cadena de texto con la URL por cargar

(este parámetro es de carácter obligatorio); y por otro, el tipo de carga que queremos hacer (método **GET** o método **POST**; por defecto, el XML se carga por medio del método **GET**, salvo que le indiquemos lo contrario). Si prestamos atención, veremos que dentro del constructor no le estamos pasando la cadena de texto **urlPath**, sino que estamos haciendo uso de una clase llamada **NoCache**.

En las próximas líneas la explicaremos:

```
xmlLoader = new XMLDocumentLoader(cadenaACargar, Método)
```

A continuación, asignamos dos listeners que contiene la clase: uno nos informará cuando se completó la carga, y otro nos informará en caso de que se produzca un error durante el proceso:

```
xmlLoader.addEventListener(Event.COMPLETE, onXmlComplete, false, 0, true);
xmlLoader.addEventListener(IOErrorEvent.IO_ERROR, onIOError, false, 0, true);
```

Finalmente, llamamos al método **loadDocument()**, que se ocupa de inicializar la carga:

```
xmlLoader.loadDocument();
```

**Figura 5.** La función **`loadDocument()`** carga la estructura XML e informa del proceso o de la ocurrencia de algun error.

# Evitar la caché del navegador: NoCache.as

La caché es la capacidad con la que cuenta una aplicación de almacenar información para su posterior uso, ya sea para ahorrar memoria o para optimizar un proceso de descarga de contenidos. Los navegadores suelen hacer uso de la caché. Si bien esto resulta útil en algunas situaciones, cuando trabajamos con estructuras XML, es común que, en caso de que realicemos alguna modificación en el archivo **.XML**, esta no se re□eje debido a que el navegador no nos muestra la última versión del archivo, sino la almacenada en la caché. Es sabido que existe un modo sencillo de evitar la caché del navegador, y es agregando alguna variable con un valor aleatorio como parámetro **GET**. Generalmente, llamamos a esta variable randomy le asignamos un número generado de manera aleatoria a la misma.

```
site.xml&random=90093973
```

Ahora bien, implementar esta solución nos traería dos problemas. Por un lado, en caso de trabajar con varios archivos XML, deberíamos asignar y quitar valores aleatorios a todos los archivos, lo cual suele resultar sumamente tedioso. Por el otro, deberíamos asignar el valor aleatorio al trabajar online, y quitarlo al hacerlo offline, ya que, si lo implementamos de manera offline, en todo caso Flash se encargará de informarnos del error que vemos a continuación:

```
TypeError: Error #1034: Error de conversión forzada: no se puede convertir
    □ash.events::Event@2702ed31 en □ash.events.IOErrorEvent.
```

## PERMITIR DEPURACIÓN DESDE FLASH

Si presionamos **CTRL+SHIFT+1** dentro de Flash, se abrirá la **Configuración de Publicación**. Al dirigirnos a la solapa **Flash**, veremos que, en el apartado **Avanzado**, hay una opción llamada **Permitir Depuración**. Cuando la activamos, cada vez que ocurra un error, se nos indicará en qué archivo y en qué línea sucedió.

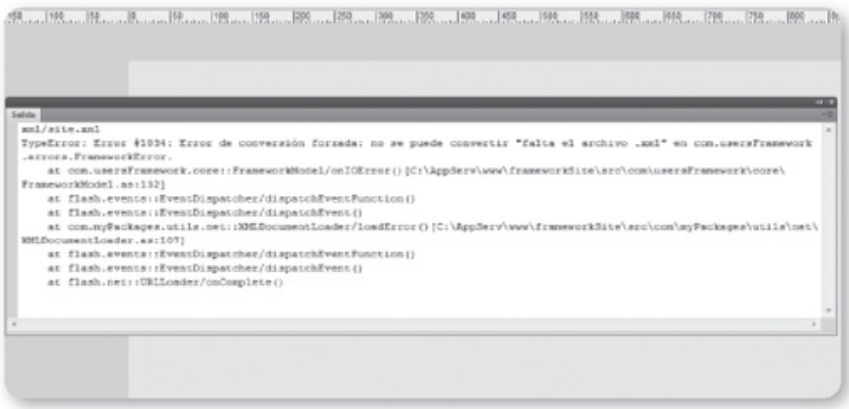



**Figura 6.** El panel **Salida** dirá que no se pudo cargar el archivo si pasamos una variable **GET** fuera de un entorno de un servidor.

Para evitar estos problemas, utilizamos la clase **NoCache.as**, que se ocupa de asignar a la URL un valor aleatorio si nos encontramos trabajando en un servidor, y de no hacerlo en caso de que estemos trabajando en local. Para que la clase pueda hacer lo suyo, precisamos pasarle dos parámetros: la URL del archivo y un valor booleano que indique si estamos trabajando en un entorno online u offline:

```
NoCache.create(path, Constants.IS_ONLINE);
```

Para que sea una mejor solución, prestemos atención a la manera de asignar el valor booleano al segundo parámetro:

```
Constants.IS_ONLINE
```

## DEBUGGING

Existen varias maneras de verificar el entorno dentro del cual nos encontramos trabajando. Independientemente de la modalidad que se emplee, lo importante es que una vez publicado el contenido evitemos posibles errores y mensajes de salida innecesarios, por esta razón es fundamental realizar un correcto debugging en nuestros proyectos.

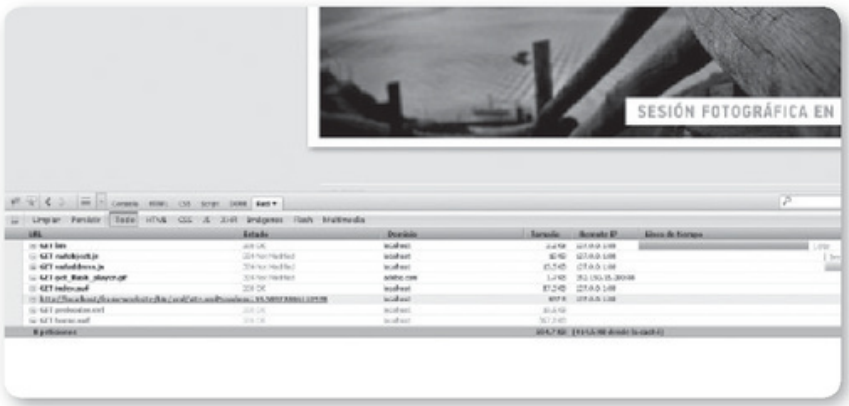



**Figura 7.** Al trabajar en un entorno web, asignamos un valor aleatorio al archivo XML. Aquí, Firebug nos muestra la URL.

## Uso de constantes: Constants.as

De nada sirve implementar esta solución si cada vez que hacemos uso de la clase **NoCache.as** tenemos que modificar manualmente el estado en el cual se encuentra la aplicación (online u offline). En estos casos, lo ideal es implementar el uso de constantes en nuestro desarrollo y tomar ese valor desde cualquier parte del código.

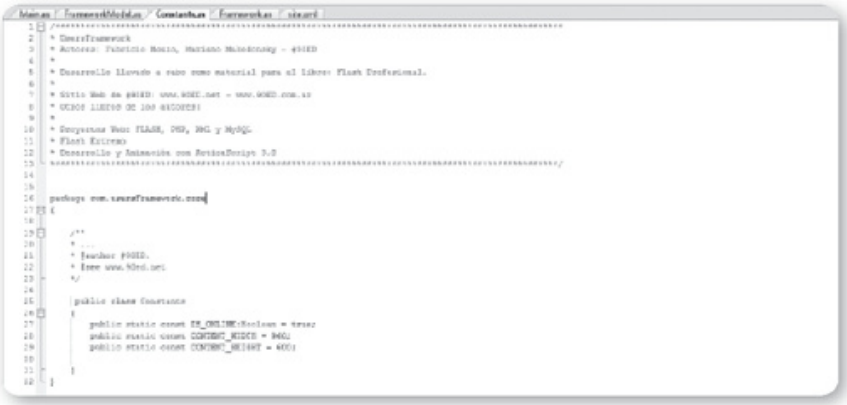


**Figura 8.** Estructura de la clase **`Constants`**. Sus variables son públicas y estáticas, y la clase no tiene constructor.

Todas aquellas constantes que admiten algún tipo de modificación durante el desarrollo se agregan a la clase **Constants.as**, veremos que realmente optimizan el tiempo de trabajo de forma considerable.

```
package com.usersFramework.core
{

    /**
    * ...
    * @author #90ED.
    * @see www.90ed.net
    */

     public class Constants
    {
        public static const IS_ONLINE:Boolean = false;
    }
}
```

De esta manera, cuando tengamos que subir nuestro proyecto, solamente deberemos modificar el valor de la variable **IS_ONLINE** a **true**, y el comportamiento de la aplicación cambiará por completo sin que tengamos que hacer más modificaciones que esa.

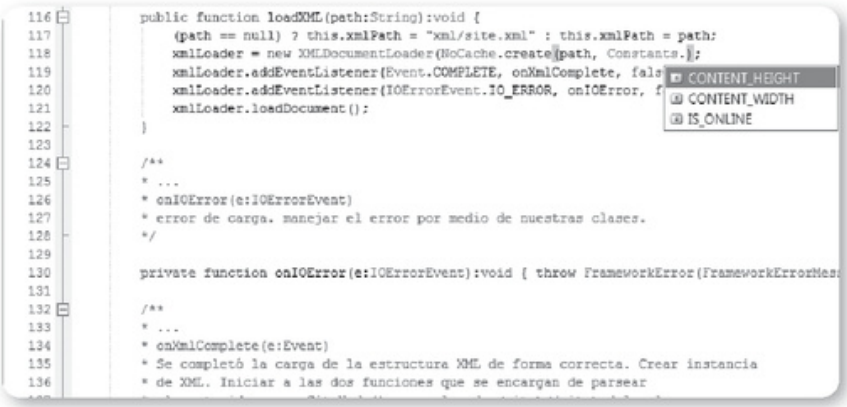



**Figura 9.** Al hacer uso de la clase **Constants**, se despliega un menú que nos muestra las constantes que esta incluye.

Resulta sumamente implementar y hacer uso de la clase NoCache. Veremos que esta cuenta únicamente con un método estático, que es el que nos brinda la funcionalidad que necesitamos.

```
public static function create(url:String, isOnline:Boolean = false):String
{
    if (isOnline)
    {
        var finalRandom:String = "random=" + Math.random() * 100;
        var finalUrl:String = (url.indexOf("?") > -1) ? url + "&" + finalRandom :
            url + "?" + finalRandom;
        return finalUrl
    }
    return url;
}
```

En primer lugar, pregunta si el parámetro **isOnline** es verdadero o falso. En caso de ser verdadero, genera un número aleatorio y se lo asigna a la variable **finalRandom**:

```
var finalRandom:String = "random=" + Math.random() * 100;
```

Por último, por medio de **indexOf()** averigua si la cadena de texto ya contiene una variable **GET**. Sobre esta base, la clase ya sabe si debe concatenar el valor aleatorio por medio del signo **?** o del signo **&**.

Para implementar la clase **XMLDocumentLoader** y la clase **NoCache** en conjunto, lo hacemos de la siguiente manera:

```
xmlLoader = new XMLDocumentLoader(NoCache.create(path, Constants.
    IS_ONLINE));
```

## VALOR ALEATORIO PARA EVITAR CACHÉ

En la clase conocida como **NoCache.as** creamos un valor aleatorio por medio de la función **Math.random()**. Tengamos en cuenta que existen otras alternativas para crear valores aleatorios: una de ellas es empleando la clase **Date()** y manipulando su información. De esta manera lo importante es crear un valor único e irrepetible para pasar como parámetro **GET** en la URL.

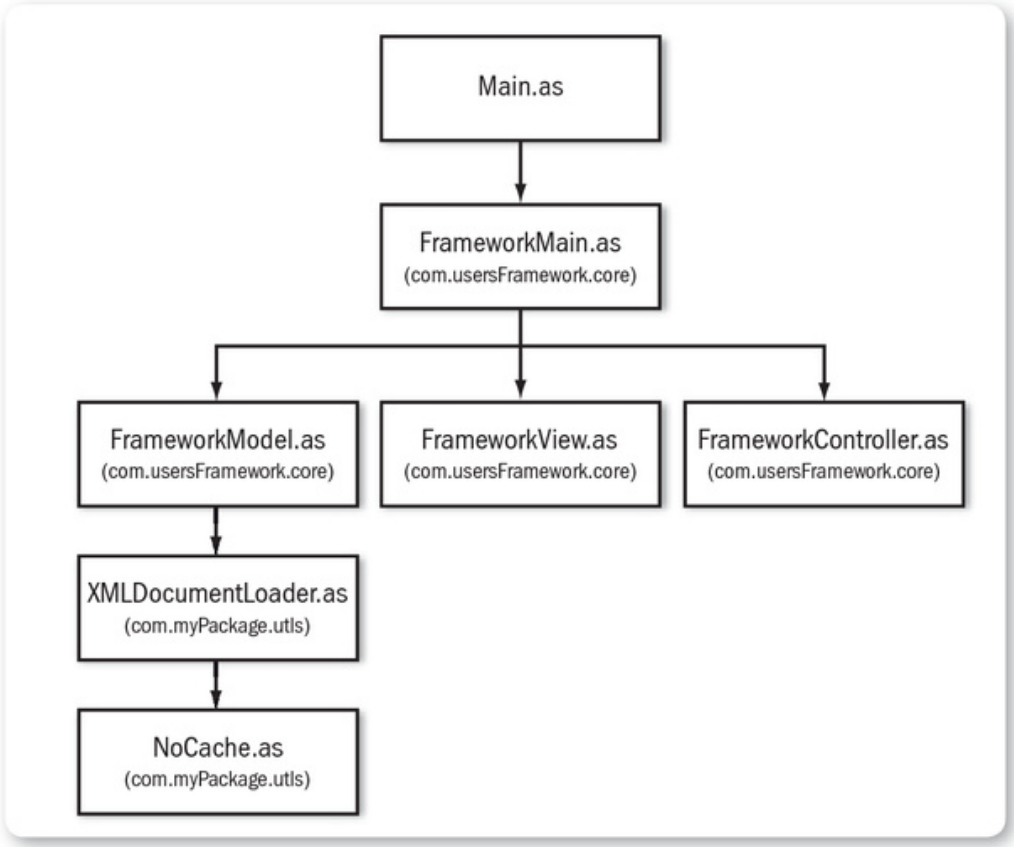



**Figura 10.** Utilizamos la clase **NoCache** dentro de la clase **XMLDocumentLoader**; tengamos en cuenta que esta carga la estructura dentro del modelo (**FrameworkModel**).

# Debugging de contenidos: Log.as

Utilizamos la clase **Log.as** para hacer *debugging* de contenidos (*logs*, *warns*, *errors*). Al igual que la clase **XMLDocumentLoader**, la clase **Log** tiene un gran alcance: si bien nos es de utilidad en este *framework*, podemos necesitarla ante cualquier tipo de proyecto: el *debug* es imprescindible durante el proceso de desarrollo de cualquier clase de aplicación.

Si bien contamos con soluciones útiles y prácticas para obtener mensajes de salida de Flash, sabemos que las aplicaciones se comportan de maneras muy diferentes dependiendo de su entorno. Por medio de la función **trace()**, podemos obtener mensajes de salida únicamente dentro de la IDE de Flash, pero no podemos leerlos al correr la película en *stand alone*, ni al encontrarnos dentro de un navegador.

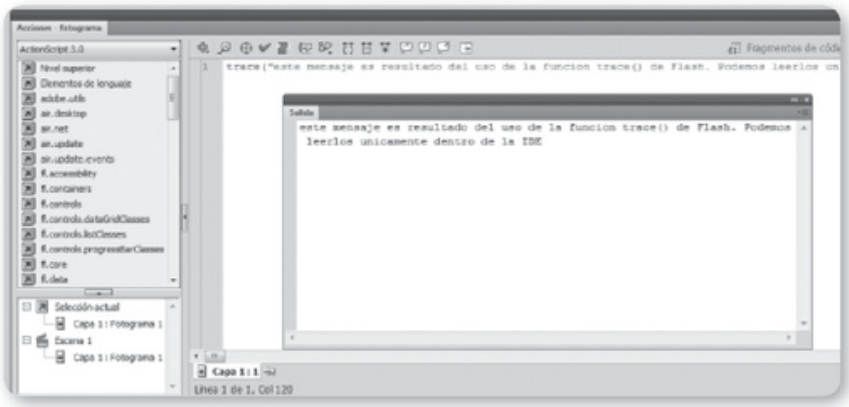



**Figura 11.** Los mensajes de salida creados con la función **`trace()`** de Flash solo pueden visualizarse dentro de la IDE.

Por otra parte, podemos utilizar **MonsterDebugger** para depurar en las tres circunstancias, pero no siempre es lo ideal: definitivamente, es mucho más sencillo, práctico y útil leer los mensajes de salida de Flash desde el panel **SALIDA** cuando estamos dentro de la interfaz gráfica de usuario de Flash. Por otro lado, contamos con la posibilidad de depurar a través de **Firebug**, pero esta solución es útil solo dentro del navegador. Ante tantas alternativas, la ventaja que tenemos es que existen soluciones para depurar código en cualquier entorno, y la desventaja es que la forma de *debuggear* es distinta en los tres casos citados.

No resulta cómodo estar modificando el código cada vez que cambiamos de entorno. Por este motivo, **veremos de qué manera implementar una solución útil y reutilizable para que, por medio de una sencilla línea de código, podamos leer los mensajes de salida independientemente del entorno**, ya sea dentro de la interfaz, en modo *stand alone*, o bien en el navegador.

## OTRAS MODALIDADES DE DEBUGGING

Si bien existen varias maneras más de hacer debug de contenidos, las que analizaremos en este libro son las que se emplean con más frecuencia: MonsterDebugger representa una gran solución a este problema, y a su vez, leer los mensajes de salida desde Firebug nos es de gran ayuda para los casos en los que necesitemos testar el contenido en el navegador web.

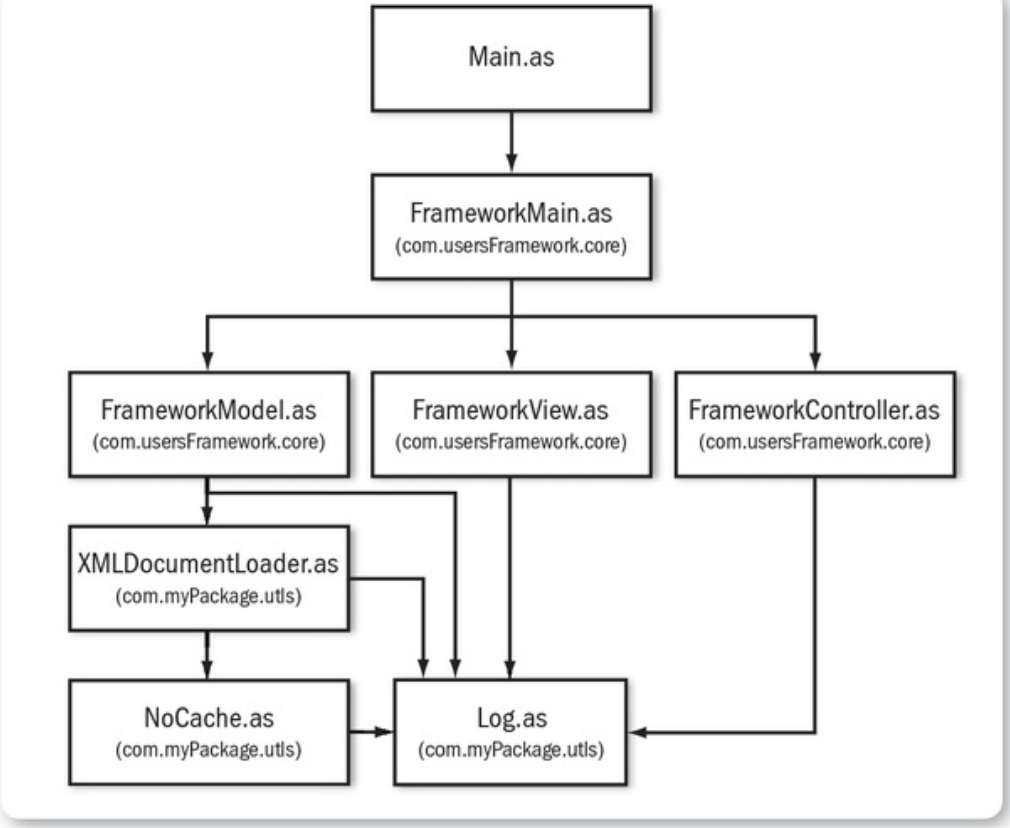



**Figura 12.** Si bien estamos explicando el uso de la clase **Log** dentro del modelo, podemos importarla y utilizarla donde la necesitemos.

Al abrir el archivo **Log.as** (**com.myPackages.debug**), veremos que la clase no cuenta con un constructor, sino con tres métodos estáticos: **log()**, **error()** y **warn()**. En primer lugar, debemos averiguar dentro de qué entorno se está ejecutando la aplicación. Tengamos en cuenta que eso se hace por medio de las variables **isBrowser** e **isConsole**.

## ¿Cuándo estamos dentro de un navegador?

Es importante saber que Flash nos brinda información como para saber cuándo el entorno de la aplicación es el navegador. Una vez que contamos con esos datos, debemos averiguar si nuestra aplicación puede comunicarse con su contenedor y si las especificaciones de seguridad correspondientes nos permiten establecer la comunicación dentro del entorno en el cual se encuentra el archivo.

## Averiguar el entorno de ejecución: Capabilities

La clase **Capabilities** nos brinda información que describe tanto el sistema como el entorno de ejecución en el cual se encuentra nuestra aplicación. Es una clase de inmensa utilidad que nos puede brindar soluciones dentro de cualquier desarrollo.

Por medio de la propiedad **playerType**, podemos averiguar el tipo de entorno dentro del cual está nuestra película. Los valores que nos devuelve son los que figuran en la **Tabla 1**.

**PROPIEDADES DE CAPABILITIES**

| PROPIEDAD | DESCRIPCIÓN |
|---|---|
| **activeX** | La aplicación se está ejecutando dentro de Internet Explorer. |
| **Desktop** | Se trata de una aplicación de Adobe AIR. |
| **External** | Se trata de una película Flash externa o bien en modo de prueba. |
| **PlugIn** | La aplicación se encuentra dentro de cualquier otro navegador (Mozilla Firefox, Google Chrome, etcétera). |
| **StandAlone** | La aplicación se está ejecutando en modo stand alone. |

**Tabla 1.** Propiedades disponibles para la clase **`Capabilities`** de Flash.

Si la propiedad **activeX** nos indica que la aplicación se está ejecutando dentro del navegador Internet Explorer, y la propiedad **PlugIn** indica que se encuentra dentro de cualquier otro navegador, sabemos que, ante cualquiera de estas dos situaciones, la película se está ejecutando dentro de un navegador web.

### CLASE CAPABILITIES

La clase **Capabilities** es verdaderamente útil: nos permite acceder a muchísima información del sistema operativo del usuario y de su versión de Flash Player. Conociendo estos datos, podemos modificar el comportamiento de nuestra película para brindarle una mejor experiencia a quien visita el sitio. Las ventajas de esta clase son inmensas y podemos emplearla para brindar las más diversas soluciones.

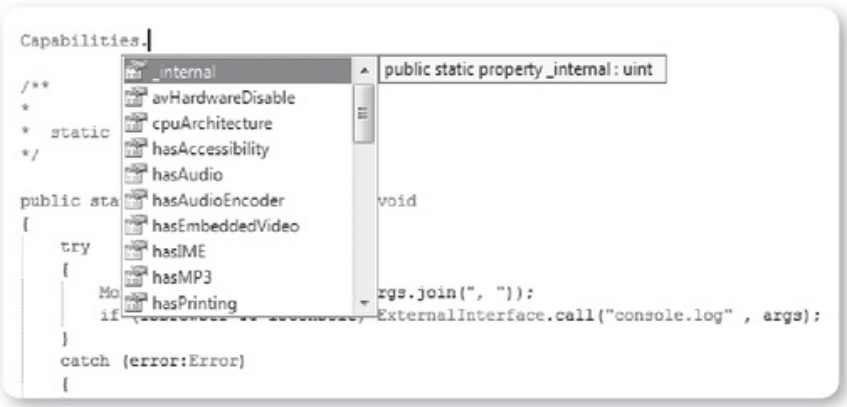



**Figura 13.** Al escribir **Capabilities** seguida de un punto, veremos la información disponible por medio de la clase.

En ese caso, igualamos la condición con la variable **isBrowser**:

```
private static var isBrowser:Boolean = (Capabilities.playerType == "ActiveX" ||
    Capabilities.playerType == "PlugIn");
```

## Comunicación entre la película y su contenedor: ExternalInterface

Con saber que la película se encuentra dentro del navegador no alcanza: debemos averiguar si esta se puede comunicar con su contenedor. Esto se realiza por medio de la clase **ExternalInterface**, que nos informa respecto a las posibilidades con las que contamos para poder comunicar a Flash ya sea una página HTML o un archivo JavaScript. Esta clase tiene una propiedad de solo lectura denominada

### SOBRE LA CLASE EXTERNALINTERFACE

Si bien empleamos la clase denominada **ExternalInterface** para brindar una utilidad de debugging a través de **Firebug**, es necesario saber que esta clase es de gran utilidad dentro del entorno Flash cuando queremos comunicar nuestra película con contenido externo, por esta razón se la suele emplear para comunicar a Flash con JavaScript cuando sea necesario.

**available**, que nos indica si nuestra película está dentro de un contenedor que nos permite establecer una comunicación externa.

```
private static var isConsole:Boolean = ExternalInterface.available
```


**Figura 14.** Dentro de **`ExternalInterface`** se encuentra la propiedad **`available`**, y el método **`call()`**.

## Comunicación dentro del dominio: clase Security

Aun así, sabiendo que estamos dentro del navegador y que podemos establecer una comunicación con el contenedor, nos falta saber si la seguridad del entorno nos permite hacerlo. Por medio de la propiedad **sandboxType** de la clase **Security**, podemos obtener el tipo de entorno de seguridad dentro del cual se encuentra nuestro archivo. Si el valor de esta propiedad es **"remote"** (**Security.REMOTE**), quiere decir que el archivo procede desde una URL de Internet; mientras que en caso de ser **"localTrusted"** (**Security.LOCAL_TRUSTED**), se trata de un archivo local, donde el usuario determinó, desde el Administrador de configuración, que es un archivo confiable. En cualquiera de los dos casos, sabemos que podemos mostrar los mensajes a través de Firebug:

```
private static var isConsole:Boolean = ExternalInterface.available && (Security.
    sandboxType == Security.REMOTE || Security.sandboxType == Security.
        LOCAL_TRUSTED);
```


**Figura 15.** Propiedades, constantes y métodos a los cuales podemos acceder a través de la clase **`Security`**.

## Poner todo a funcionar en conjunto

Veamos de qué manera ejecutamos los mensajes de salida dependiendo del entorno en el que estamos. Utilizaremos como ejemplo la función **log()**. Las funciones **warn()** y **error()** son exactamente iguales, salvo que, en vez de realizar la función a **"console.log"** de Firebug, lo harán a **"console.warn"** y a **"console.error"**, respectivamente:

```
public static function log(...args):void
{
    try
    {
        MonsterDebugger.trace(Log, args.join(", "));
        if (isBrowser && isConsole) ExternalInterface.call("console.log" , args);
    }
catch (error:Error)
    {
    }
    trace.apply(null, args);
}
```

La sentencia **try..catch..finally** nos es útil en aquellos casos en los que queremos incluir un bloque de código que puede llegar a generar algún

error, y si se detectar dicho error, podemos responder a él. Nuestros mensajes de salida los incluiremos dentro de esta sentencia. Veamos los resultados que obtenemos si ejecutamos la siguiente línea una vez que se carga la estructura XML:

```
Log.log("xml cargado correctamente");
```

## Debug a través de MonsterDebugger

Debemos tener en cuenta que por medio de la propiedad denominada **trace()** de la clase **MonsterDebugger**, indicamos el *target* y los mensajes que queremos mostrar. En este caso, los separamos con una coma por medio de la función **join()**:

```
MonsterDebugger.trace(Log, args.join(", "));
```

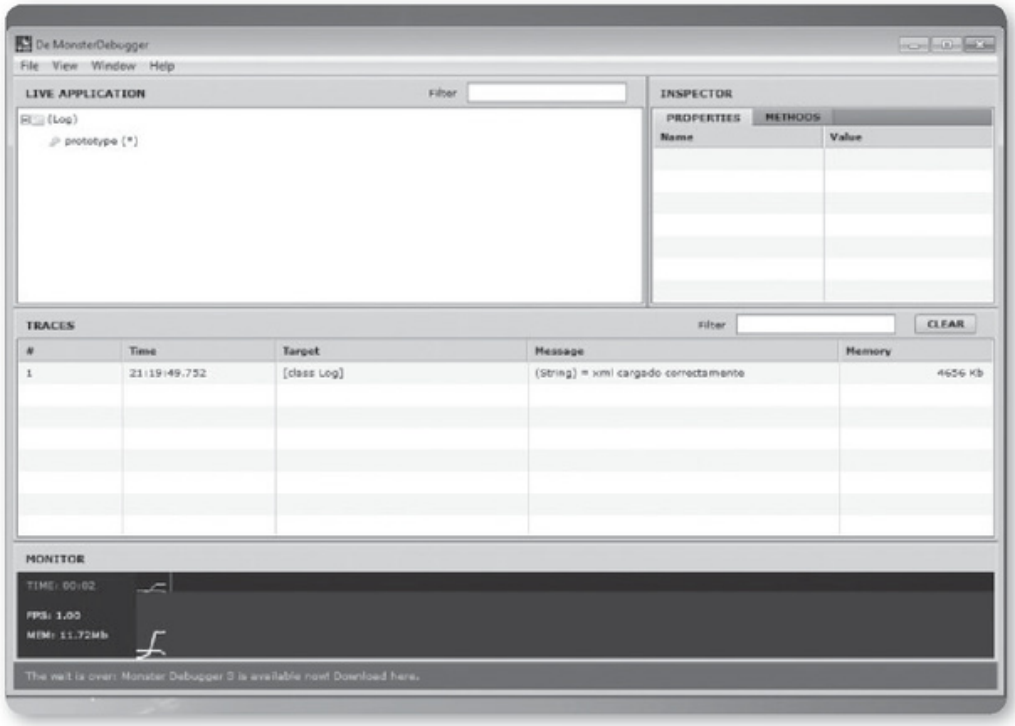



**Figura 16.** Mensaje de salida que podemos ver desde la interfaz de usuario que nos presenta **MonsterDebugger**.

## Debug a través de Firebug

Anteriormente hicimos las validaciones pertinentes para depurar por medio del add-on Firebug. En caso de darse las dos condiciones (**isBrowser** e **isConsole**), hacemos uso del método **call()** de la clase **ExternalInterface**. Este método nos permite llamar a una función que se encuentre dentro del contenedor de nuestra película, pasándole los argumentos que correspondan:

```
if (isBrowser && isConsole) ExternalInterface.call("console.log", args);
```

En nuestro caso, nos encargamos de realizar la ejecución de la función **"console.log"** de Firebug, y le pasamos como argumentos la variable **args**, que contiene nuestro mensaje de salida.

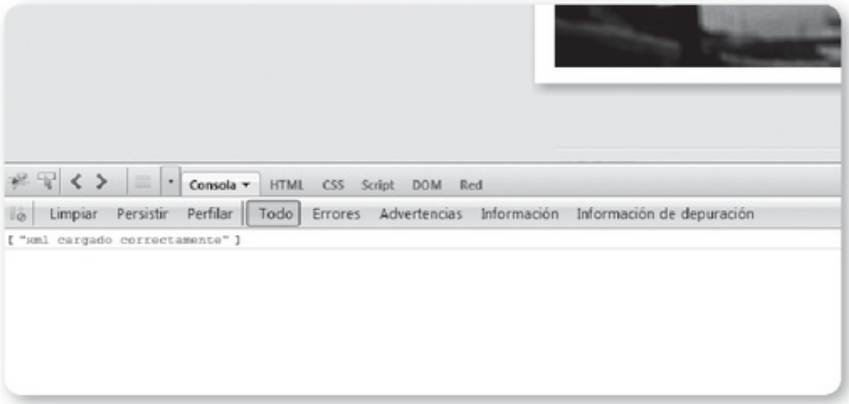



**Figura 17.** El mismo mensaje de salida, pero en esta ocasión, visto desde la interfaz propia de Firebug.

## Debug dentro de Flash

Por último, hacemos el trace dentro de Flash por medio de la función **trace**. Esta es la misma función trace() que conocemos desde siempre de Flash, salvo que en este caso, ejecutamos el mensaje haciendo uso del método apply() para pasarle los argumentos.

```
trace.apply(null, args);
```

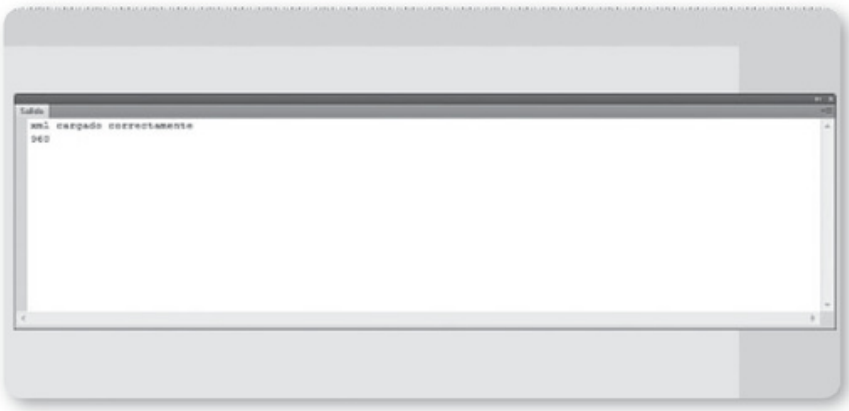


**Figura 18.** Por último, en esta imagen podemos ver el mensaje de salida dentro de la interfaz de Flash.

A partir de ahora, cada vez que queramos ejecutar algún mensaje de salida, ya sea de información, de advertencia o de error, simplemente importamos la clase **Log.as** y lo hacemos de la siguiente manera por medio de sus métodos:

```
Log.log("un mensaje");
Log.warn("una advertencia");
Log.error("un mensaje de error");
```

¿El resultado? Veremos el mensaje de salida dentro de Flash, en la consola de MonsterDebugger y dentro del navegador, empleando una única línea para imprimirlo. De esta forma nos olvidamos por completo del entorno dentro del cual se ejecuta la aplicación y sabemos que siempre tenemos acceso a los mensajes de salida. realmente útil, ¿no?.

## ATAJO DE TECLADO DESDE FLASHDEVELOP

Si presionamos la tecla **F1** al estar posicionados sobre cualquier tipo de dato que corresponda a AS3, se realizará una búsqueda directamente en Google sobre el elemento en cuestión. Por ejemplo, si hacemos clic sobre la palabra **Boolean** y presionamos **F1**, se efectuará la siguiente búsqueda: "actionscript 3.0" Boolean Boolean site:help.adobe.com; posteriormente veremos los resultados.

# Manejo de errores

Es normal que, ante estas estructuras, nos encontremos con errores. Flash tiene un sistema de numeración mediante el cual maneja en tiempo real los errores (*runtime error handling*). En verdad, estos mensajes de error no siempre nos son de utilidad.

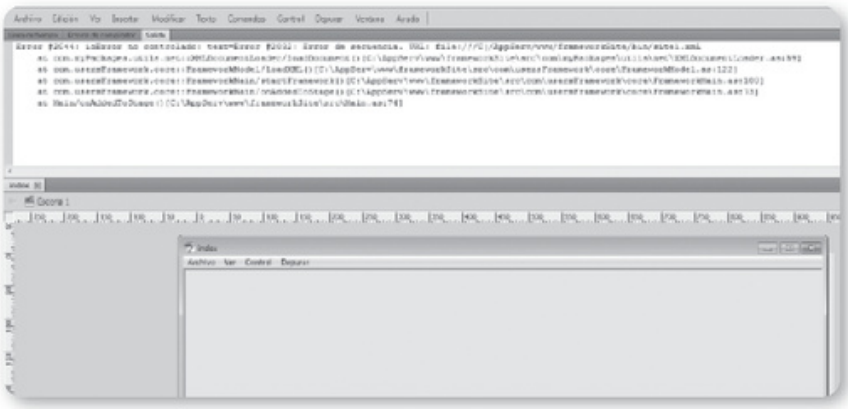


**Figura 19.** Ejemplo de error de Flash. En este caso, el error 1006. Esta información no siempre resulta útil o clara.

A la instancia de la clase **XMLDocumentLoader**, le asignamos el siguiente evento para saber cuándo se produjo un error en la carga:

```
xmlLoader.addEventListener(IOErrorEvent.IO_ERROR, onIOError, false, 0, true);
```

Al detectarse un error, podemos crear nuestro propio gestor de errores y manejarlo con información que sea más representativa del problema que se está presentando, de la siguiente forma:

## ADOBE FLASH PLAYER RUNTIME ERRORS

Es importante tener presente que desde la siguiente página podemos acceder al listado de posibles errores que pueden ocurrir en tiempo de ejecución dentro del entorno del Flash Player: **http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/runtimeErrors.html**. Los datos de esta página son de suma utilidad para estar preparados frente a las dificultades en el desarrollo.

```
throw new Error("mensaje de error");
```

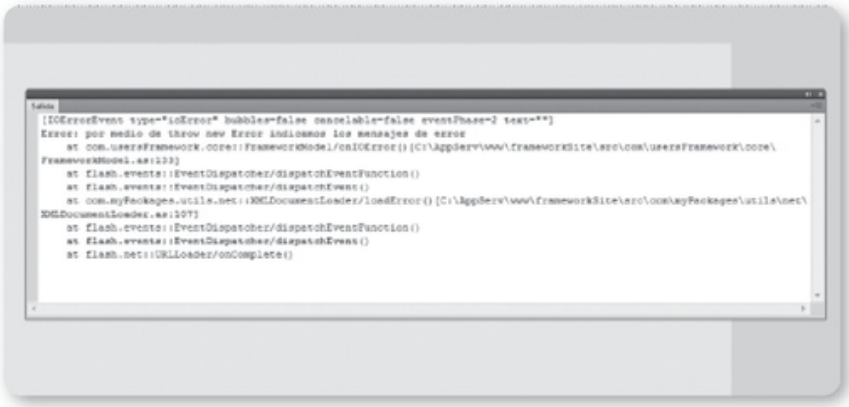



**Figura 20.** Ejemplo de error ejecutado por medio de **`throw new Error()`**.

## FrameworkError.as

La clase **FrameworkError** se ocupa de mostrar los errores de nuestra aplicación. En vez de llamar a la clase **Error** a través del constructor **New**, lo haremos de la siguiente manera:

```
new FrameworkError("mensaje de error");
```

## FrameworkErrorMessage.as

Así como la clase **Constants** tiene constantes comunes a todo el proyecto, la clase **FrameworkErrorMessages** tiene constantes para cada uno de los errores que pueden surgir en nuestro *framework*:

```
package com.usersFramework.errors{
    /**
 *
 * @author #90ED
    * @see http://www.90ed.net/
```

```
    */
      public class FrameworkErrorMessage {

          // XML
          public static const XML_PAGE_MISSING_ID:String = "no se puede
              parsear una nodo <page> de XML que no tenga su correspondiente id";
          public static const XML_PAGE_MISSING_SRC:String = "falta indicar el
              atriburo 'src' dentro del nodo <page> de la estructura XML";
          public static const XML_MISSING:String = "falta el archivo .xml";

          // LOADING
          public static const URL_FAILED:String = "No existe el archivo SWF que
              se está intentando cargar";
      }
  }
```

Desde luego que nuestro espacio de trabajo seguiría siendo 100% funcional en ausencia del gestor de errores que hemos visto en este capítulo, pero el propósito de este libro no es únicamente generar un espacio de trabajo, sino además brindar información respecto a buenas prácticas por implementar que hacen de nuestro desarrollo un mejor entorno. Este tipo de prácticas es particularmente útil cuando los proyectos crecen de modo considerable, y también para tener un mejor flujo de trabajo en grupos interdisciplinarios.

**Figura 21.** En esta imagen vemos algunos de los errores que controlamos mediante nuestro gestor de errores.

# Parseo de la estructura XML

Si retomamos las páginas anteriores, veremos que a la instancia de la clase **XMLDocumentLoader** le asignamos el siguiente evento:

```
xmlLoader.addEventListener(Event.COMPLETE, onXmlComplete, false, 0, true);
```

Este evento se ejecutará cuando la estructura XML se encuentre dentro de nuestra película, y en ese caso, se va a ejecutar la función **onXmlComplete()**:

```
private function onXmlComplete(e:Event):void {
    _siteXml = xmlLoader.data;
    parseSiteNode();
    parsePagesNodes();
}
```

En primer lugar, igualamos la variable **_siteXml** a la propiedad **data** de nuestra instancia de la clase **XMLDocumentLoader** (**xmlLoader.data**). A partir de ahora, la variable **_siteXml** contiene la estructura XML que acabamos de cargar. A continuación, nos ocuparemos de llamar a las funciones **parseSiteNode()** y **parsePagesNodes()**.

**Figura 22.** Estructura XML sobre la base de la cual tenemos la información para crear y gestionar el espacio de trabajo.

## Parseo del nodo <site> del XML: función parseSiteNode()

Dentro de la función **parseSiteNode()**, lo que hacemos es igualar las variables que tenemos declaradas dentro de la clase **FrameworkModel.as** con los atributos que contiene el nodo **site** del documento XML:

```
/**
* ...
* parseSiteNode()
* parseo de nodo site y asignación de sus valores a variables privadas y
* estáticas de la clase.
*/
private function parseSiteNode():void {
    //titulo del sitio
    _siteTitle = _siteXml.@title || "";
    //menu contextual
    _siteContextMenu = (_siteXml.@menu == "true");
    //delimitador
    _siteDelimiter = _siteXml.@delimiter || " // ";
    //accesibilidad
    _siteAccessibility = (_siteXml.@accessibility == "true");
   //preloader
```

```
    _sitePreloader = _siteXml.@preloader || "preloader.swf";
    //FPS
    _siteFrameRate = int(_siteXml.@framerate) || 24;
    //fullScreen
    _siteFullScreen = (_siteXml.@fullScreen == "true");
    //defaultPage
    _siteDefaultPage = (_siteXml.@defaultPage);
}
```


**Figura 23.** Información que nos encargamos de almacenar dentro de la función **`parseSiteNode()`** del modelo.

## Parseo de los nodos <page> del XML: parsePagesNodes()

La gran diferencia entre el nodo **site** y el nodo **page** del XML es que **site** hay solamente uno, en tanto que **page** puede haber un número indefinido. Es por este motivo que debemos darles un tratamiento diferente a estos últimos:

```
/**
* ...
* parsePagesNodes()
* parseo de los nodos <page> del site del XML
*/
```

```
private function parsePagesNodes():void{
    _siteMenu = new Vector.<Object>
    var i:uint;
    var length:uint = _siteXml.*.length();
    for (i = 0; i < length; i++) {
    _siteMenu.push(readPageNodeData(_siteXml.child(i)));
    }
    if (_siteLandingObject == null) _siteLandingObject = _siteMenu[0];
    dispatchEvent(new Event(Event.COMPLETE));
}
```

En primer lugar, creamos un vector dentro del cual vamos a almacenar un objeto por cada página que contenga la estructura XML:

```
_siteMenu = new Vector.<Object>
```

Y a continuación, recorremos la estructura XML con un bucle **for**:

```
var i:uint;
var length:uint = _siteXml.*.length();
for (i = 0; i < length; i++) {
    _siteMenu.push(readPageNodeData(_siteXml.child(i))
    ;
}
```

Si prestamos atención, al agregar un elemento al vector (**_siteMenu.push()**), estamos haciendo uso de la función **readPageNodeData()**.

## VENTAJA DE VECTOR SOBRE ARRAY

Al almacenar los objetos con la información de cada página, lo hacemos sobre un vector y no, sobre un array. Cuando los elementos por agregar son del mismo tipo de dato (en nuestro caso son todos objetos), es preferible utilizar vectores, ya que su acceso y manipulación son más rápidos.

## Análisis del contenido: readPageNodeData()

Más importante que analizar línea por línea la función **readPageNodeData()** es comprender su objetivo. La función recibe por parámetro un nodo de la estructura XML (**node:XMLList**), y nos devuelve un objeto (**return pageObject**). De esta manera, cada vez que el bucle **for** que creamos realiza una vuelta, sucede lo siguiente:

- Se le pasa a la función como parámetro un nodo **page** (**node:XMLList**).
- Para continuar se valida que la información proporcionada por el nodo sea correcta y que no falte ningún atributo obligatorio a través de la función denominada **validatePageNode()**.
- Se crea un objeto (**pageObject**) y se le asignan variables, a las cuales se iguala con la información que contiene el nodo del XML.
- La función devuelve un objeto con esta información (**pageObject**).

```
/**
* ...
* readPageNodeData(node:XMLList)
* @param node nodo del XML sobre el cual queremos obtener la informacion
* @return pageObject : objeto que contiene la informacion del nodo
*/

private function readPageNodeData(node:XMLList):Object {
    validatePageNode(node);
    var pageObject:Object = new Object();
    pageObject.id = node.@id;
    pageObject.src = node.@src;
    pageObject.urlFriendly = node.@urlFriendly;
    pageObject.landing = (node.@src == "true");
    pageObject.menu = (node.@menu == "true");
    pageObject.external = (ExtensionManager.getExtension(node.@src) ==
        "swf") ? true : false;

    if (pageObject.external) pageObject.window = node.@window || "_self";
    if (pageObject.id == _siteDefaultPage) _siteLandingObject = pageObject;
    return pageObject;
}
```

De esta manera, cada vez que se realiza una vuelta del ciclo **for**, se almacena, dentro del vector, un objeto con la información de página:

```
_siteMenu.push(readPageNodeData(_siteXml.child(i))
```

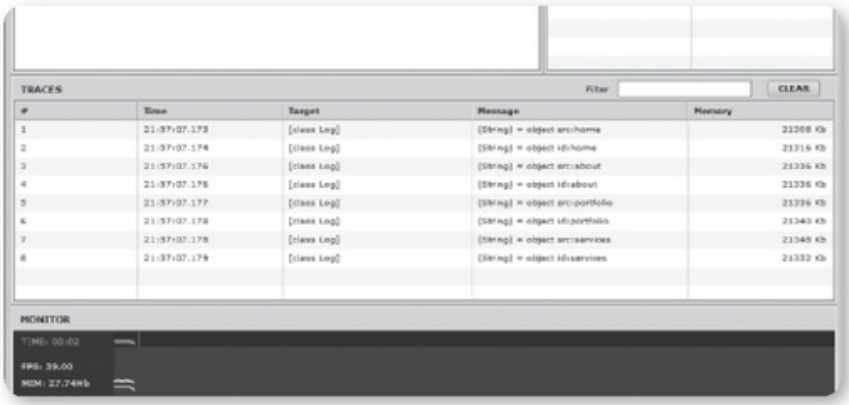



**Figura 24.** Log de la información que se va almacenando en cada objeto vista desde MonsterDebugger.

## Validación del contenido: validatePageNode()

En el **Capítulo 3** enumeramos los atributos que les corresponden al nodo **site** y a los nodos **page** del XML. Algunos de los atributos de los nodos **page** tienen que ser de carácter obligatorio, porque sin ellos, no podemos hacer funcionar el espacio de trabajo. Tal es el caso del atributo **src** (el cual indica la ruta del archivo SWF por cargar), y del atributo **id** (identificador de página). Estas validaciones se llevan a cabo de la forma que ejemplificamos en el siguiente código:

### FLASHDEVELOP: @PARAM

En toda función que se encarga de recibir parámetros, si antes de declararla escribimos a modo de comentario **@param** seguido del nombre del parámetro y una descripción de este, cuando hagamos uso de la función, al abrir paréntesis, podremos acceder a la información que seteamos: el parámetro por pasar y una descripción sobre él. Hacer esto es de inmensa utilidad.

```
/**
* ...
* validatePageNode(node:XMLList)
* @param node - validacion de nodo <page> para atributos obligatorios
*/

private function validatePageNode(node:XMLList):void {
if (node.@src == undefined) {
        throw FrameworkError(FrameworkErrorMessage.XML_PAGE_
            MISSING_SRC);
    }else if (node.@id == undefined) {
        throw FrameworkError(FrameworkErrorMessage.XML_PAGE_
            MISSING_ID);
    }
}
```

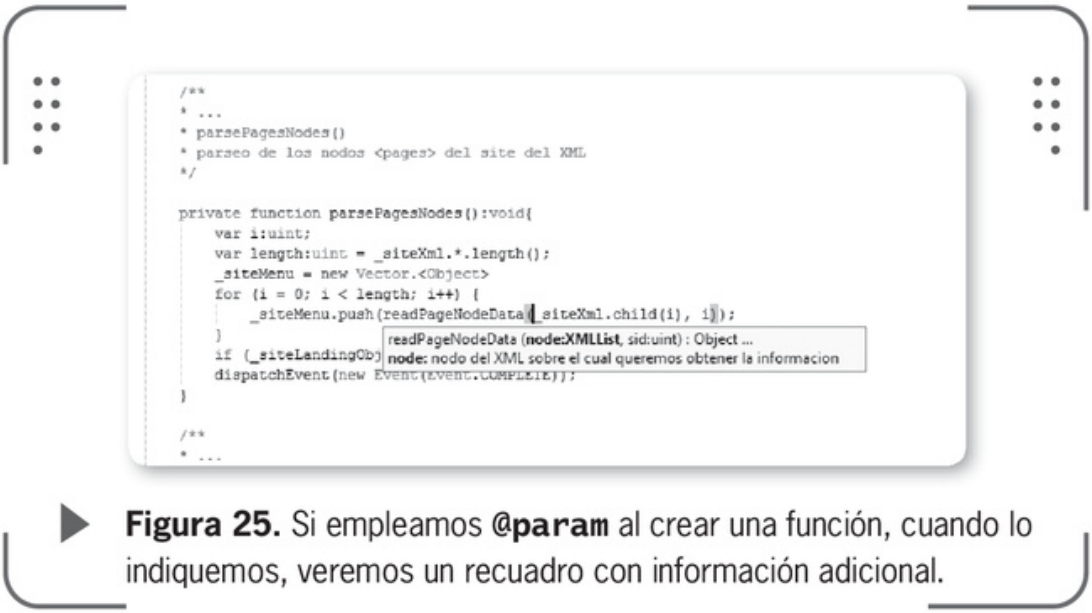



**Figura 25.** Si empleamos **@param** al crear una función, cuando lo indiquemos, veremos un recuadro con información adicional.

## FLASHDEVELOP: @RETURN

En aquellas funciones que devuelven algún valor, al escribir a modo de comentario **@return**, podemos indicar qué valor devuelve y una descripción sobre él. De este modo, cuando escribimos el nombre de la función y abrimos paréntesis, se nos mostrarán los datos que indicamos para ella.

Si prestamos atención a la clase, veremos que estamos haciendo uso nuevamente de nuestro gestor de errores, en caso de que tengamos que informar que está faltando un atributo obligatorio.

# Encapsulación en el modelo: getters y setters

Otro concepto importante que vale la pena repasar respecto a la Programación Orientada a Objetos es el de encapsulación, y el modelo de nuestro proyecto es un gran ejemplo de esto. La encapsulación nos permite manejar y definir información de objetos sin necesidad de conocer su funcionamiento interno. Se suele definir este comportamiento como **caja negra**: en verdad, no sabemos cómo es la estructura interna del objeto que creamos (en este caso, lo sabemos porque acabamos de crear nosotros mismos el modelo, pero esto no tiene por qué ser importante a los fines prácticos). Lo que nosotros hicimos previamente fue almacenar información, y lo que tenemos que hacer a continuación es lograr que esa información sea accesible para el resto de las clases que componen nuestro entorno de trabajo. Puesto de manera sencilla, a partir de aquí, no nos interesa cómo se cargó el archivo XML. Tampoco importa de qué manera evitar la caché, ni cómo gestionar errores. Lo que sí nos interesa es poder acceder a la información para leerla o modificarla con el uso de **getters** y **setters**.

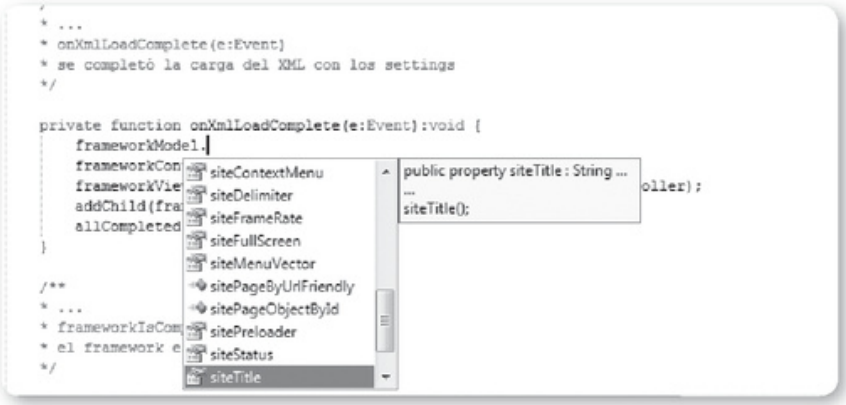



**Figura 26.** Al emplear `FrameworkModel`, tenemos acceso al listado de getters y setters que definimos para ella.

## Getters y Setters: obtener y setear información

Los *getters* y *setters* nos permiten acceder a las propiedades y métodos de un objeto sin necesidad de conocer su funcionamiento.

Dentro de la clase **FrameworkModel**, creamos *getters* y *setters* para toda la información que hemos almacenado. Los *getters* permitirán el acceso a esta información desde otras clases. Veamos algunos ejemplos:

Código dentro de **FrameworkModel.as**:

```
/**
* ...
* siteTitle();
* @return _siteTitle : devuelve el titulo del sitio
*/

public function get siteTitle():String
{
    return _siteTitle;
}

/**
* ...
* siteContextMenu()
* @return _siteContextMenu : booleano que indica si el sitio cuenta con menu
    contextual
*/

public function get siteContextMenu():Boolean
{
   return _siteContextMenu;
```

### GETTER Y SETTER AUTOMÁTICAMENTE

Una vez declarada una variable no pública, si presionamos **CTRL+SHIFT+1** mientras estamos posicionados sobre ella, FlashDevelop nos dará la posibilidad de generar un *getter* y un *setter* automáticamente para dicha variable. Este es un *shortcut* muy útil que nos permite ahorrar escritura de código.

```
}

/**
* ...
* siteMenuVector()
* @return _siteMenu : obtener el vector con el menu del sitio
*/

public function get siteMenuVector():Vector.<Object> {
    return _siteMenu;
}
```

Estos tres *getters* nos permiten acceder al nombre, al título de nuestro sitio, a un *booleano* que indica si el sitio tiene un menú contextual y a otro que nos permite obtener el vector que contiene toda la información de las páginas del sitio. Para obtener esta información desde cualquier otra clase, lo hacemos de la siguiente manera:

```
var title:String = FrameworkModel.getInstance().siteTitle;
var contextMenu:Boolean = FrameworkModel.getInstance().siteContextMenu;
var vector:Vector = FrameworkModel.getInstance().siteMenuVector;
```

Y los *setters* nos permitirán definir información:
Código dentro de **FrameworkModel.as**:

```
/**
* ...
* siteFrameRate(frameRate:int)
* @param frameRate : setter para modificar el framerate
*/

public function set siteFrameRate(frameRate:int):void
{
    _siteFrameRate = frameRate;
}
```

En este caso, podríamos modificar el valor de la variable **_siteFrameRate** desde cualquier otra clase, de la siguiente manera:

```
FrameworkModel.getInstance().siteFrameRate = 32
```

## Getters y setters: métodos descriptores de acceso o funciones

Tenemos dos formas de crear *getters* y *setters*. Una es hacerlo como si se tratase de métodos, de la siguiente forma:

```
/**
* ...
* siteFrameRate(frameRate:int)
* @param frameRate : setter para modificar el framerate
*/

public function siteFrameRate(frameRate:int):void
{
    _siteFrameRate = frameRate;
}
```

La otra modalidad es por medio de los métodos descriptores de acceso **get** y **set** que nos ofrece ActionScript 3.0:

```
/**
* ...
* siteFrameRate(frameRate:int)
* @param frameRate : setter para modificar el framerate
*/

public function set siteFrameRate(frameRate:int):void
{
    _siteFrameRate = frameRate;
}
```

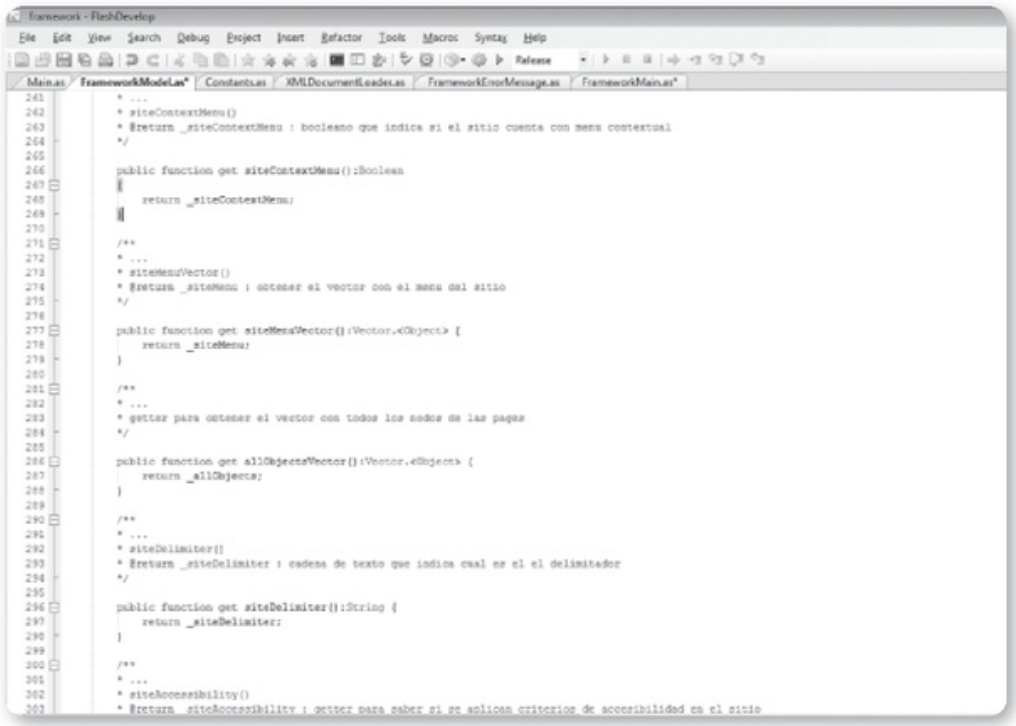



**Figura 27.** En nuestro caso, empleamos descriptores de acceso tanto para obtener valores como para definirlos dentro del modelo.

Si comparamos las dos fracciones de código, vemos que la única diferencia que hay entre la función y el método descriptor de acceso es la anteposición en el segundo caso del *key* **set** antes del nombre de la función. La ventaja del uso de descriptores de acceso es que podemos tener un único nombre de función, para *settear* información y obtenerla. Miremos el *getter* para el *frame rate*, y veremos que el nombre de la función es exactamente igual al nombre de función que lleva el *setter*:

```
/**
* ...
* siteFrameRate()
* @return _siteFrameRate : obtener el framerate de la pelicula
*/
public function get siteFrameRate():int {
    return _siteFrameRate;
}
```

La diferencia que presenta el uso de funciones o el uso de descriptores de acceso es la modalidad en la cual captamos o definimos información. Cuando se está empleando una función, accedemos a la información como si se tratase de un método:

```
FrameworkModel.getInstance().getSiteFrameRate();
```

Mientras que al utilizar un descriptor de acceso, la utilizamos como una propiedad:

```
FrameworkModel.getInstance().siteFrameRate;
```

O bien, como dijimos anteriormente, en este caso podemos utilizar el mismo nombre de función tanto para captar como para definir información. En el caso del *getter* también lo hacemos mediante su propiedad, como podemos ver en el siguiente código, en el cual asignamos la velocidad de fotogramas por segundo de nuestra película.

```
FrameworkModel.getInstance().siteFrameRate = 24;
```

# Fin del almacenamiento de la información

Si volvemos a la clase **FrameworkMain**, veremos, una vez más, de qué manera armamos el Singleton para nuestra clase **FrameworkModel**:

## ALCANCE DEL MODELO

Debemos tener en cuenta que si bien la información que estamos almacenando en el modelo se corresponde con un espacio de trabajo orientado al desarrollo de sitios web, la manera de pensarlo como clase que carga la información y la hace accesible al resto de las clases resulta útil para cualquier tipo de desarrollo. El modelo es una forma transparente de manejar datos.

```
/**
* ...
* startFramework()
* función que da inicio al framework y crea el MVC
*/

private function startFramework():void {
    frameworkModel = FrameworkModel.getInstance();
    frameworkModel.addEventListener(Event.COMPLETE, onXmlLoadComplete,
        false, 0, true);
    frameworkModel.loadXML(siteXML);
}
```

Nos interesa particularmente la siguiente línea:

```
frameworkModel.addEventListener(Event.COMPLETE, onXmlLoadComplete,
    false, 0, true);
```

Cuando se ejecute la función **onXmlLoadComplete()**, quiere decir, nada más y nada menos, que se concluyó la carga del modelo. En ese caso, debemos inicializar el controlador y la vista:

```
/**
* ...
* onXmlLoadComplete(e:Event)
* se completó la carga del XML con los settings
*/

private function onXmlLoadComplete(e:Event):void {
    frameworkController = new FrameworkController(frameworkModel);
    frameworkView = new FrameworkView(frameworkModel,
        frameworkController);
    addChild(frameworkView);
    frameworkIsCompleted();
}
```

```
/**
* ...
* onXmlLoadComplete(e:Event)
* se completó la carga del XML con los settings
*/

private function onXmlLoadComplete(e:Event):void {
    frameworkModel.
    frameworkController = new FrameworkController(frameworkModel);
    frameworkView = new FrameworkView(frameworkModel, frameworkController);
    addChild(frameworkView);
    allCompleted();
}

/**
* ...
* frameworkIsCompleted()
* el framework está completo
*/

protected function allCompleted():void {
    framework = new Framework();
    stage.frameRate = framework.getFrameRate();
    if (framework.getContextMenu()) contextMenu = FrameworkContextMenu.init(framework.getFullScreen());
    if (framework.getAccessibility()) FrameworkAccessibility.init(stage);
    TweenMax.delayedCall(1, init);
```

**Figura 28.** Una vez concluida la carga del modelo, procedemos a levantar la vista y el controlador de nuestro espacio de trabajo.

Para continuar, nos resta inicializar la vista y el controlador. En el próximo capítulo nos abocaremos de lleno al controlador.

# Vista del framework: FrameworkView.as

Como mencionamos al comenzar el capítulo, a la vista le corresponde la visualización de los contenidos implementados y la interacción dentro de la tríada conocida como MVC.

En nuestro caso puntual, y considerando que se trata de un *framework*, nosotros llamaremos vistas, en verdad, a los contenedores que se ocuparán de lo que mencionamos a continuación:

## OTROS PATRONES DE DISEÑO

Es necesario tener en cuenta que si bien hasta aquí hemos visto el patrón MVC y el Singleton, existe una larga lista de patrones de diseño que se pueden emplear dependiendo de los objetivos del desarrollo al cual nos estemos enfrentando, entre ellos encontramos los siguientes: **adapter**, **decorator**, **interpreter**, **iterator**, **mediator**, **observer**, **state**, **strategy**, entre otros.

- Cargar y descargar las páginas del sitio.
- Almacenar el menú.
- Cargar y descargar el preloader.

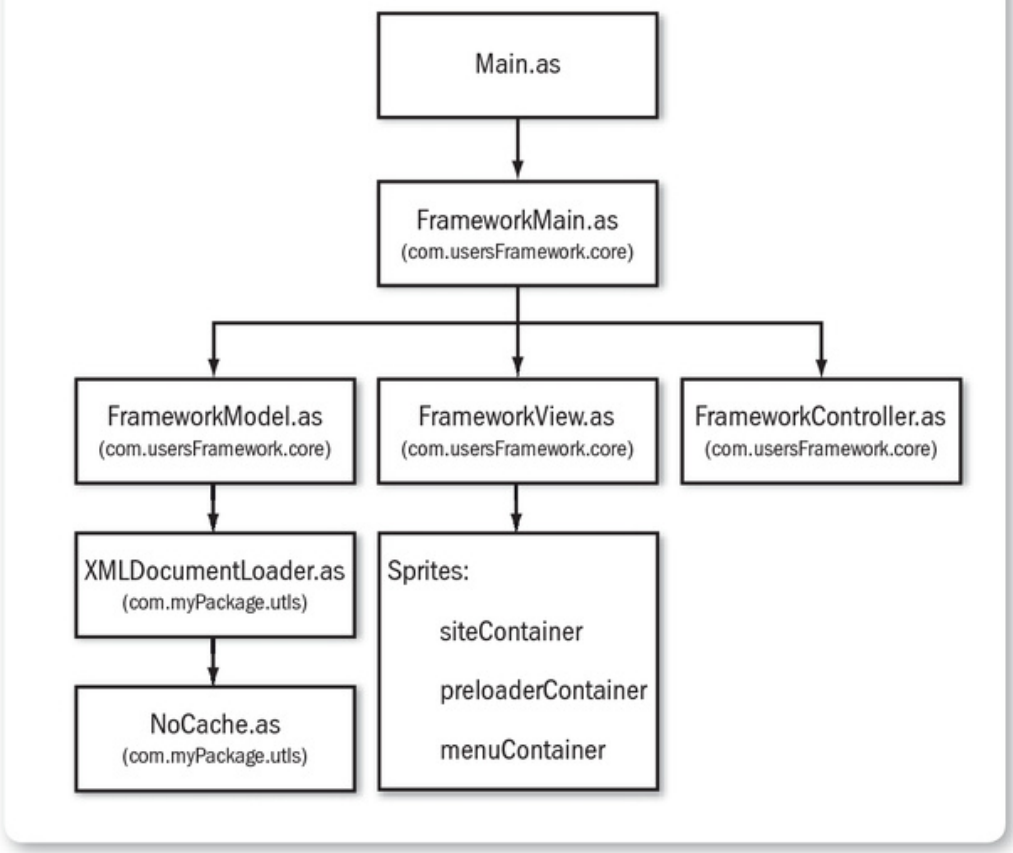



**Figura 29.** Desde la vista manejamos los elementos visuales del desarrollo y los disponemos en el escenario adecuado.

# Interactividad en el framework e interactividad en las páginas

Es muy importante diferenciar estos dos conceptos: **un elemento es el framework en sí, y otro son las páginas que componen un desarrollo cuando hacemos uso del framework**.

Lo que vamos a abarcar en este capítulo está vinculado a la creación del *framework*, y dentro de él, los únicos elementos que tendrán

interacción son los botones de la botonera y los del menú contextual. Distinto es el caso de las páginas que componen un sitio. ¿Qué quiere decir esto? Que el MVC que estamos desarrollando actualmente es parte de un espacio de trabajo: reiteramos innumerables veces que un espacio de trabajo es la creación de un entorno reutilizable, y dentro de él, consideramos que es necesaria la creación de una mínima forma de interacción. Ahora bien, cada página tiene su propio nivel de interacción, que puede estar dado de la manera que a uno se le ocurra: ya sea por elementos en la línea de tiempo, por medio de OOP y patrones de diseño, o por medio de cualquier tipo de interacción que Flash nos permita usar. Esto no nos interesa y no es nuestro tema de desarrollo: la interacción que forma parte de cada página corre por cuenta de cada desarrollador. Esto implica que podemos estar usando un MVC para manejar el *framework*, y a la vez, cada sección del sitio puede contar con un MVC o con varios más. Lo que nos incumbe en este libro es cómo manejar la estructura principal.

# Crear la vista

Hagamos un pequeño repaso de lo que es la vista: dentro de ella, el usuario interactúa y visualiza los contenidos. Al comienzo del capítulo dijimos que es en la vista donde se da el *input* (interacción del usuario). Esta información debe enviarse al controlador, el cual realizará una acción según la información que reciba, y luego, le informará al modelo que cambie su estado. Cuando el modelo lo haga, ejecutará los cambios que sean necesarios en la vista. Es por este motivo que, al crear la clase **FrameworkView**, debemos pasarle como parámetros el modelo de nuestro sitio y el controlador: a uno lo usará para informarle de los cambios (controlador), y a otro, para asignarle eventos que modifiquen la visualización de los contenidos (modelo).

## SPRITE, MOVIECLIP Y PESO DE LAS CLASES

En casi todos los casos en que debemos extender una clase a un elemento visual, lo hacemos a **Sprite** y no a **MovieClip**. Estos dos elementos visuales son prácticamente iguales, pero **Sprite** no contiene línea de tiempo, y esto hace que sea unos pocos bytes más liviano.


**Figura 30.** A la vista del sitio debemos pasarle instancias, del modelo y controlador, ya que tendrá que interactuar con ambas.

## FrameworkView.as

A diferencia del modelo, que contiene y manipula información y, por ese motivo, extiende a la clase **EventDispatcher**, en el caso de la vista se trata de un elemento visual y necesitamos añadirla al escenario. Por eso precisamos que extienda a **Sprite** o **MovieClip**:

```
public class FrameworkView extends Sprite {
```

Ya dentro del constructor de la clase **FrameworkView**, recibimos los parámetros y los igualamos a variables internas para manipularlas dentro de cualquier lugar de la clase. A continuación, creamos los tres contenedores que mencionamos anteriormente, y asignamos un evento para saber cuándo se añadió la vista al escenario:

### EXTENDIENDO A SPRITE

Es importante señalar que la tarea de extender a Sprite en vez de hacerlo a MovieClip reduce el peso final de nuestra película. Dedicaremos un capítulo entero a performance, y dentro del mismo analizaremos esta situación con detenimiento. Por ahora basta comprender que depende de las decisiones que tomemos en la etapa de desarrollo será el resultado de rendimiento de la aplicación final.

```
public function FrameworkView(frameworkModel:FrameworkModel, framework
    Controller:FrameworkController):void {
super();
    this.frameworkModel = frameworkModel;
    this.frameworkController = frameworkController;
    siteContainer = new Sprite();
    preloaderContainer = new Sprite();
  menuContainer = new Sprite();
    addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
}
```

Una vez que se ejecuta el evento **Event.ADDED_TO_STAGE**, sabemos que nuestra vista se encuentra en el escenario. En primer lugar, agregamos los contenedores al escenario:

```
/**
* ...
* onAddedToStage()
* una vez que se agrega al escenario, agregar los listeners
*/

private function onAddedToStage(event:Event):void {
    removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
    addChild(siteContainer);
    addChild(preloaderContainer);
    addChild(menuContainer);
```

## ADDCHILD() Y ADDCHILDAT()

Como sabemos, por medio del método denominado **addChild()** podemos encargarnos de añadir un **display object** a la **display list**. Ahora bien, si queremos indicar un índice para definir la profundidad, podemos utilizar el método llamado **addChildAt(child, index)**. Es importante tener esto en cuenta ya que seguramente utilizaremos estos métodos en nuestros desarrollos.

# Cambio de estado: del modelo a la vista

Esta es la parte más sencilla de la vista. La forma de interacción que tendrá el usuario se verá en el próximo capítulo al centrarnos en la **usabilidad** y **accesibilidad** de nuestro espacio de trabajo. Lo que nos interesa ahora es explicar de qué manera asignar eventos al modelo para que se ejecuten dentro de la vista. **Recordemos que el modelo almacena la información y modifica el estado de la aplicación**. Ahora, ¿qué implica modificar el estado de la aplicación? Si bien el espacio de trabajo presenta cierta complejidad, este no tiene muchos estados: podemos diferenciarlos claramente en cuatro:

- El modelo le pide a la vista que agregue una página.
- El modelo le pide a la vista que quite una página.
- El modelo le pide a la vista que agregue el preloader del sitio.
- El modelo le pide a la vista que quite el preloader del sitio.

```
frameworkModel.addEventListener(FrameworkModel.ADD_PAGE, onAddPage,
    false, 0, true);
frameworkModel.addEventListener(FrameworkModel.REMOVE_ACTUAL_
    PAGE, onRemoveActualPage, false, 0, true);
frameworkModel.addEventListener(FrameworkModel.ADD_PRELOADER,
    onAddPreloader, false, 0, true);
frameworkModel.addEventListener(FrameworkModel.REMOVE_PRELOADER,
    onRemovePreloader, false, 0, true);
```

Cada uno de estos listeners ejecuta su respectiva función:

## PACIENCIA Y PERSEVERANCIA

Es muy importante recordar que lograr ☒uidez en el desarrollo de aplicaciones creadas bajo el paradigma de Programación Orientada a Objetos suele requerir años de desarrollo y experiencia. Por lo tanto, no debemos desalentarnos ante la complejidad que esta tarea presenta, sino que debemos ser perseverantes y pacientes en el aprendizaje, y de esta forma obtendremos excelente resultados.

```
/**
* ...
* addPage();
*/

public function onAddPage(e:Event):void {
    var page = FrameworkModel.getInstance().pendingPage;
    siteContainer.addChild(page);
}

/**
* ...
* addPreloader(preloader)
*/

public function onAddPreloader(e:Event):void {
    var page = FrameworkModel.getInstance().preloaderPage;
    preloaderContainer.addChild(page);
}

/**
* ...
* removeActualPage(page)
* @param page : página actual a eliminar
*/

public function onRemoveActualPage(e:Event):void {
    var page = FrameworkModel.getInstance().actualPage;
   siteContainer.removeChild(page);
```

## SHORTCUT DE FLASHDEVELOP

Los *snippets* son fracciones de código que habitualmente necesitamos insertar y utilizar dentro de nuestro desarrollo: por ejemplo, un ciclo **for**, un **switch** o un **if**. Presionando **CTRL+B**, se desplegará este listado de acciones. Luego, si seleccionamos cualquiera de ellas, se generará automáticamente el código.

```
}

/**
* ...
* removePreloader(preloader)
* @param preloader : el preloader preloader a eliminar
*/

public function onRemovePreloader(e:Event):void {
    var page = FrameworkModel.getInstance().preloaderPage;
    preloaderContainer.removeChild(page);
}
```

Este es uno de los puntos más complejos de nuestro entorno de trabajo. Acostumbrarse a hacer uso de OOP y de patrones de diseño generalmente lleva tiempo y dedicación. En nuestro caso, la mejor alternativa para entenderlo es abrir los archivos de ejemplo que hemos creado y analizarlo detenidamente. Lo que sí, indudablemente, una vez que comprendemos la lógica de esta modalidad de desarrollo, veremos las ventajas y una inmensa cantidad de soluciones a diversos problemas.

## RESUMEN

En este capítulo vimos los conceptos básicos que giran en torno al uso de OOP y patrones de diseño en nuestro framework. La ventaja de estructurar el código de esta manera es que, en caso de querer efectuar alguna modificación o implementación, resulta sencillo hacerlo y no tenemos la necesidad de involucrarnos de modo innecesario en clases que cumplan con otra funcionalidad. A su vez, vimos varios ejemplos de clases que nos brindan soluciones útiles ante diversos casos, para así optimizar los tiempos de desarrollo.

# Actividades

## TEST DE AUTOEVALUACIÓN

| | |
|---|---|
| 1 | ¿Cuál es la ventaja de utilizar patrones de diseño? |
| 2 | ¿En qué casos resulta conveniente hacerlo? |
| 3 | ¿Qué es el patrón MVC? |
| 4 | ¿De qué se encarga el modelo dentro de un patrón MVC? |
| 5 | ¿De qué se ocupan la vista y el controlador dentro de un patrón MVC? |
| 6 | ¿Qué es un patrón Singleton? ¿Cuáles son sus características? |
| 7 | ¿Cuál es la ventaja de utilizar constantes dentro de un proyecto? |
| 8 | ¿De qué maneras podemos debuggear una aplicación? |
| 9 | ¿Qué es la encapsulación dentro de OOP? ¿Cuáles son sus características? |
| 10 | ¿Para qué sirven los métodos descriptores de acceso **set** y **get**? |

## ACTIVIDADES PRÁCTICAS

| | |
|---|---|
| 1 | Cree un ejemplo sencillo utilizando un patrón MVC. |
| 2 | Genere los getters y setters para la información con la que cuenta el modelo. |
| 3 | Acceda a la información que brindan los getters y setters desde cualquier otra clase. |
| 4 | Cree un ejemplo cualquiera utilizando el patrón Singleton. |
| 5 | Haga pruebas en local y en un navegador para ver el comportamiento de la caché. Utilice la clase vista en este capítulo o cree una nueva para evitarlo. |

*06

# Usabilidad y accesibilidad

Flash cuenta con una serie de recursos que son de inmensa utilidad a la hora de brindar una mejor experiencia de usuario. En este capítulo veremos algunas de estas soluciones, la manera de implementarlas dentro de nuestro entorno, y una serie de conceptos que nos servirán de aquí en adelante ante cualquier tipo de desarrollo.

# Ideas sobre usabilidad y accesibilidad

Uno de los principales problemas con los que tuvo que lidiar la Plataforma Adobe Flash desde sus inicios fue su mala reputación tanto en SEO como en usabilidad y accesibilidad. Generalmente, culpar a una tecnología es eximirnos de nuestras responsabilidades como desarrolladores. Uno de los pilares sobre los cuales sostenemos nuestra manera de encarar cada desarrollo es considerando que no existen buenas o malas tecnologías, sino buenas o malas implementaciones de ellas, y buenas o malas elecciones al momento de elegir la herramienta adecuada. Flash presenta una gran cantidad de recursos para hacer que cualquier aplicación resulte más usable o más accesible, pero la usabilidad y la accesibilidad son mucho más que código: hay una gran parte de la responsabilidad que recae sobre nosotros, sobre nuestros conocimientos y nuestros recursos. **La usabilidad, independientemente del entorno en el cual se implementa, se basa en cuán eficiente, agradable e intuitiva es la experiencia que proponemos para un usuario**.

Para lograrlo, gran parte recae sobre la manera en la cual implementamos el código de nuestros proyectos, y otra gran parte, sobre conceptos que hacen al diseño de interfaces.

## Diferenciar la usabilidad de la accesibilidad

Se suele confundir la usabilidad con la accesibilidad. La mejor manera de entender la diferencia es por medio de un sencillo ejemplo:

**DISEÑO DE INTERFACES**

Lamentablemente, no podemos abarcar en este libro conceptos avanzados sobre diseño, ya que escapa de nuestro objetivo, pero no hay que perder de vista que muchos de los criterios de usabilidad no forman parte del mundo de la programación, sino del mundo del diseño en comunicación visual.

si un edificio cuenta con una escalera mecánica, ese edificio está siendo usable; está brindando una mejor y más eficiente experiencia al usuario. Ahora bien, un discapacitado motriz no puede hacer uso de ella. La escalera mecánica hace al edificio usable, pero no lo hace accesible. Para que sea accesible, necesitaría tener, además, un ascensor. **Mientras que la usabilidad se basa en brindar una mejor experiencia, la accesibilidad implica que cualquier persona pueda utilizar algo, independientemente de sus capacidades técnicas, físicas o cognitivas**.

# Alcance de los conceptos

En lo que respecta a muchos de los conceptos desarrollados en este libro, nos es imposible abarcarlos en su totalidad, considerando que tenemos un único capítulo para cada uno de ellos. Cada uno de estos temas, tratado en profundidad, podría llevar varios tomos de libros similares a este. Tenemos dos objetivos para este capítulo: por un lado, dotar de recursos usables y accesibles al entorno de trabajo que venimos manejando; y por otro, brindar información útil para cualquier tipo de desarrollo: el hecho de pensar de manera usable y accesible es ventajoso ante cualquier tipo de desarrollo.

En este capítulo veremos:

- Cómo hacer más usable el menú de navegación del sitio.
- Cómo dotarlo de accesibilidad.
- Cómo plantear otros modos de navegación.
- Cómo podemos acceder a brindar posibilidad de navegación desde el menú contextual incorporado en Flash.
- De qué forma es posible brindar a los usuarios la opción de navegar empleando shortcuts del teclado.
- Navegación con deep linking usando SWFAddress.

## ACCESIBILIDAD Y USABILIDAD

Hay un punto interesante por considerar en el ejemplo del edificio: el ascensor no es únicamente útil para personas con discapacidades: por el contrario, cualquiera puede hacer uso de él. Al pensar en la accesibilidad, obtenemos una ventaja en la usabilidad.


**Figura 1.** En el borde superior derecho se encuentra el menú sobre el cual vamos a aplicar recursos de accesibilidad y usabilidad.

# Todo está relacionado

Si bien en este capítulo veremos una forma práctica de emplear criterios de usabilidad y accesibilidad al menú de nuestro espacio de trabajo, todo lo que hemos visto hasta ahora (y vamos a ver en próximos capítulos) hace, directa o indirectamente, a la usabilidad y a la accesibilidad. Un error que tiende a cometerse, es pensar que la usabilidad concluye con la experiencia del usuario. Si bien la experiencia de usuario es lo que determina si alcanzamos nuestras metas o no, el hecho de generar un entorno reutilizable y escalable, como hemos visto en los primeros capítulos, determina que las aplicaciones que creemos sean de fácil actualización y nos brinden la flexibilidad requerida para ofrecer esa mejor experiencia de la que hablamos. A su vez, generar contenido alternativo, como vimos en el **Capítulo 4**, no es únicamente útil para lograr buenos resultados en lo que hace a la optimización de contenidos para navegadores: también es ofrecer una solución a aquellas personas que no pueden visualizar

películas SWF, al darles otra manera de acceder a la información. Por otra parte, dedicaremos un capítulo entero a la optimización de contenidos y performance: como resultado de llevar a cabo estas buenas prácticas, lograremos sitios más fluidos, de menor tamaño, con menor consumo de memoria y mejor rendimiento. Indudablemente, sin necesidad de una repercusión visual, todo esto que mencionamos, generado desde el código, propone una mejor experiencia para el usuario y forma parte, también, de la usabilidad y la accesibilidad.

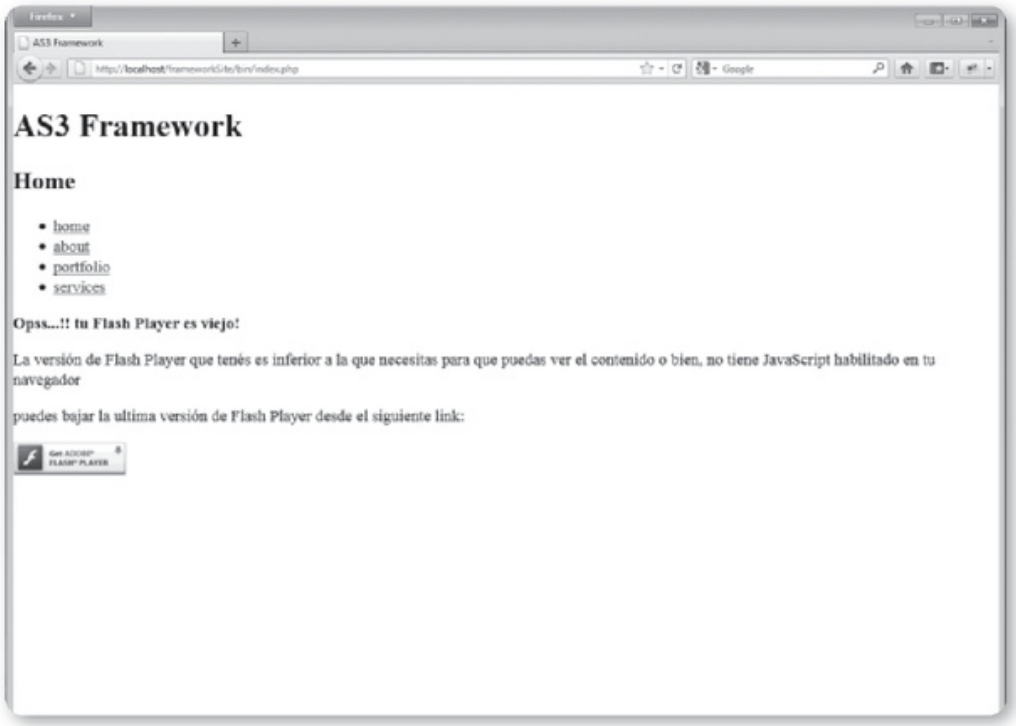



**Figura 2.** El contenido alternativo creado con **`SWFObject`** en el **Capítulo 4** contribuye a generar un sitio más usable y accesible.

## VIDEOS ÚTILES SOBRE ACCESIBILIDAD

En el sitio que se encuentra en la dirección **www.adobe.com/accessibility/products/flash/tutorial** es posible acceder a una serie de videos sobre datos útiles y consejos para crear sitios accesibles en Flash. Si queremos comenzar a profundizar en el tema, estos videos son un muy buen punto de partida. Muchos de los conceptos están desarrollados en este capítulo.

# Navegación usable

Es necesario tener en cuenta que podemos aplicar criterios de usabilidad y accesibilidad prácticamente a cualquier elemento multimedia: desde formularios de contacto hasta reproductores de audio y de video, galerías de imágenes, etcétera. Nosotros aplicaremos estos criterios a la creación del menú de navegación del sitio. El archivo sobre el cual vamos a trabajar se ubica en la carpeta **com.project.display**, y lleva por nombre **Menu.as**.

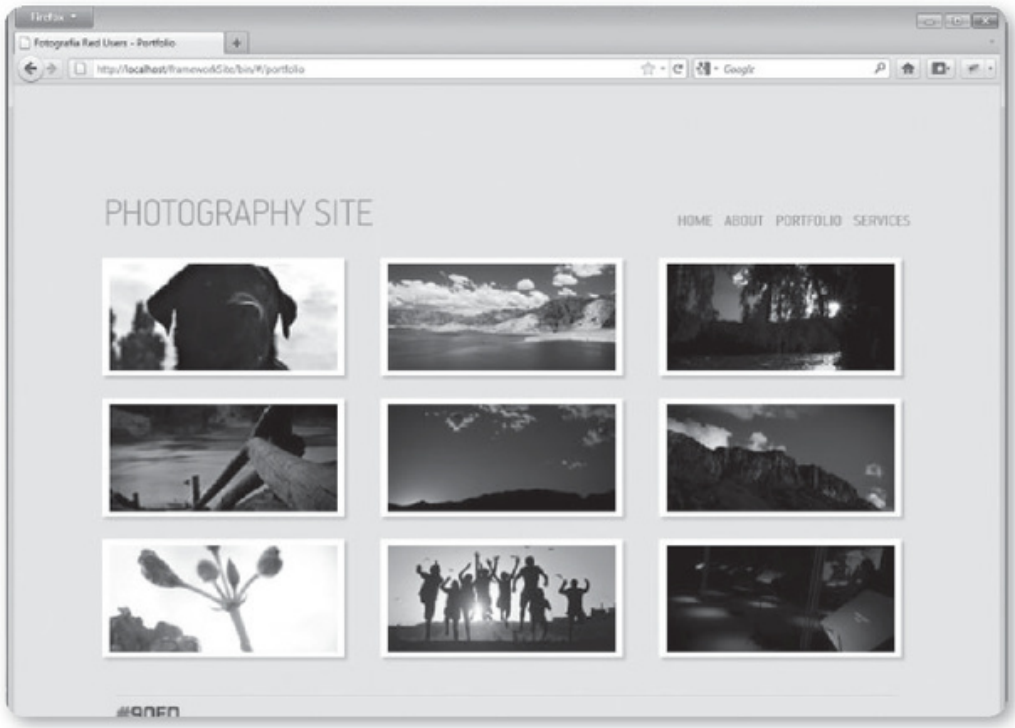



**Figura 3.** El menú de navegación presenta tres estados: la sección visitada, la sección que aún no ha sido visitada y la sección activa.

## OBSERVACIÓN DE SITIOS Y APLICACIONES

Una buena práctica es acostumbrarse a evaluar sitios, para saber cuán usables y accesibles son. Nos sorprenderemos ante la cantidad de información que este relevamiento puede brindarnos, al encontrar tanto buenos recursos como malos recursos que tendrían una sencilla solución.

# Indicar el estado de la navegación

Desde la clase **Main.as** creamos el menú de la siguiente manera:

```
Menu.getInstance().init();
addChild(Menu.getInstance());
```

Si prestamos atención, veremos que se trata de un Singleton. Hay una explicación para esto: **independientemente de que se pueda acceder a las secciones desde el menú del sitio**, también podremos hacerlo desde el menú contextual de Flash y desde el teclado. Es por esto que, sin importar el lugar desde el cual indiquemos la sección del sitio por visitar, debemos apuntar siempre hacia la misma clase a fin de modificar los estados de las secciones.

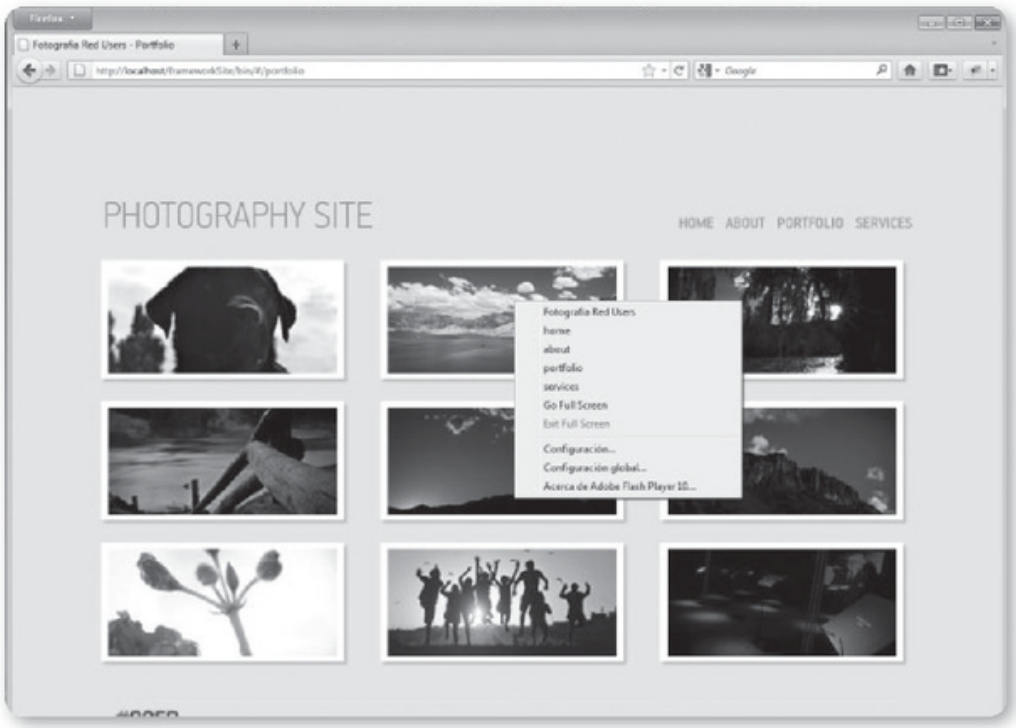



**Figura 4.** Veremos de qué manera podemos crear distintas alternativas para navegar el sitio, como se muestra en esta imagen.

## Indicar la sección actual

Este sencillo recurso no siempre se tiene en cuenta a la hora de implementar un menú de navegación. La explicación seguramente recaiga en el hecho de que lograr

una verdadera optimización requiere de tiempo y dedicación. Pero es preciso que el usuario siempre sepa en qué sección se encuentra. Con tan solo un cambio cromático, alcanza para lograr esto.

En primer lugar, debemos definir un color para cada uno de los estados: de esta manera, vamos a indicar el botón que se encuentra activo, el que fue visitado y el que aún no fue visitado:

```
private static const ACTIVE:uint = 0x15A3D7;
private static const VISITED:uint = 0x999999;
private static const UNVISITED:uint = 0x333333;
```

Y luego realizamos un bucle **for** para ir creando los distintos botones. Sabemos que la información de estos se toma del XML que parseó la clase **FrameworkModel**: de ella obtenemos el vector con los objetos y el largo del vector. La ventaja de trabajar el menú de esta manera es que, en caso de querer agregar una nueva sección, no tenemos que preocuparnos por el menú:

```
var i:uint;
var length:Number = FrameworkModel.getInstance().siteMenuVector.length;
var vector:Vector.<Object> = FrameworkModel.getInstance().siteMenuVector;
var fontX:Number = 0;
buttonsVector = new Vector.<MovieClip>();
for (i = 0; i < length; i++) {
```

## Asignar eventos y propiedades a los botones

De los eventos, indudablemente, el más importante es el que empleamos para detectar el clic sobre el botón:

```
buttonContainer.addEventListener(MouseEvent.CLICK,
onClickItem, false, 0, true);
```

Al detectarse el evento, llamamos a la función **onClickItem()**. Recordemos lo que mencionamos al principio: la navegación se llevará

a cabo desde tres lugares distintos. Es por esto que, desde aquí, solamente tomamos el id del botón y se lo pasamos a la función **setActive()**. Esta es la función principal de nuestro código. Analicemos las líneas más importantes.


**Figura 5.** No debemos olvidarnos de definir la propiedad **buttonMode** en **true** al trabajar con cualquier elemento cliqueable.

Lo primero que hacemos es validar si el id actual (**actualId**) es igual al último que ingresó a la función. Esto lo hacemos porque, si bien en el menú vamos a desactivar el botón sobre el cual se haga clic, no hay que perder de vista que llamaremos a esta función desde otras clases: podemos estar en la sección **HOME**, y presionar el *shortcut* que nos lleva también a la sección **HOME**. No tiene sentido que permitamos esto, así que, en caso de darse esta condición, salimos de la función, si lastId es igual a actualId, salimos de la función correspondiente:

```
public function setActive(actualId:uint):void {
if (lastId == actualId) return;
```

Al botón que corresponda a la sección actual, debemos eliminarle todos sus eventos e indicar que, actualmente, no se debe comportar como un botón. Esto es muy útil al considerar la modalidad en la que funcionan los lectores de pantalla (**screenreaders**) para los que necesiten soluciones accesibles, tal como vemos a continuación:

```
buttonsVector[actualId].removeEventListener(MouseEvent.CLICK, onClickItem);
buttonsVector[actualId].removeEventListener(MouseEvent.MOUSE_OVER,
    overItem);
buttonsVector[actualId].removeEventListener(MouseEvent.MOUSE_OUT,
    outItem);
buttonsVector[actualId].buttonMode = false;
buttonsVector[actualId].useHandCursor = false;
```

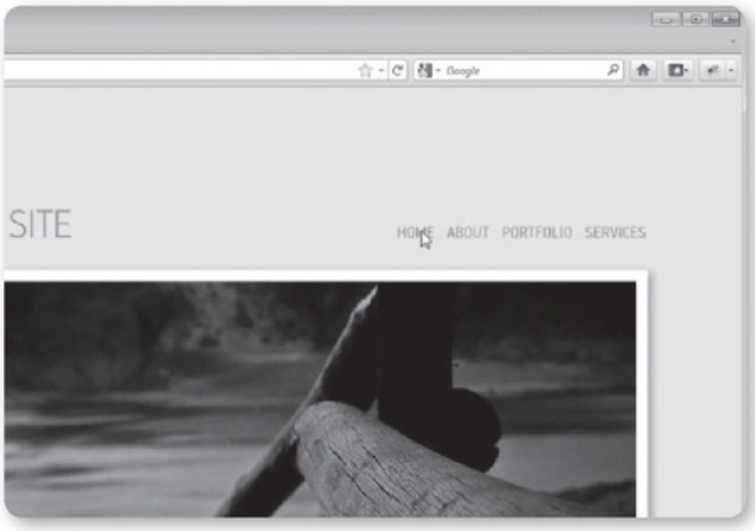



**Figura 6.** Sabemos que al estar sobre una sección, el botón que corresponda a ella debe dejar de comportarse como tal.

## CLASE SHAREDOBJECTFLUSHSTATUS

La clase **SharedObjectFlushStatus** nos permite saber si la información se almacenó correctamente o no. En caso de que nos devuelva **FLUSHED**, el almacenamiento fue exitoso, y si nos devuelve **PENDING**, significa que la operación falló. Debemos tener en cuenta que con esta información podemos controlar el correcto funcionamiento de la clase SharedObject.

## Indicar que una sección fue visitada

HTML, generalmente, hace esto por nosotros, y es una gran ventaja. No es que no podamos hacerlo en Flash; al contrario, resulta muy sencillo indicar que una sección ha sido visitada, pero no suele hacerse. Simplemente, modificamos la propiedad **visited** del MovieClip que ha sido cliqueado, de **false** a **true**:

```
if (!buttonsVector[actualId].visited) buttonsVector[actualId].visited = true;
```

## Indicar la sección actual

Ya habiendo eliminado todos los *listeners*, solo nos resta indicar de algún modo cuál es la sección actual. Por medio de la clase **TweenMax**, generamos una pequeña animación y modificamos el tinte del botón. Como color, asignamos la variable **ACTIVE**. Si recordamos, definimos esta variable al comienzo del código. Por medio de dicho color, indicamos cuál es la sección actual en la que se encuentra el usuario:

```
TweenMax.to(buttonsVector[actualId], 0.3, { tint:ACTIVE } );
```

Por último, si no es la primera vez que se cliquea un elemento del menú, quiere decir que, anteriormente, había otro botón activo. Recordemos que a este elemento lo identificamos con la variable **lastId**. Lo que hacemos es volver a asignarle todos sus eventos y comportamientos al botón que deja de ser la sección actual:

```
if (!fistClick) {
buttonsVector[lastId].addEventListener(MouseEvent.CLICK, onClickItem, false,
    0, true);
buttonsVector[lastId].buttonMode = true;
buttonsVector[lastId].useHandCursor = true;
TweenMax.to(buttonsVector[lastId], 0.3, { tint:VISITED} );
}
    lastId = actualId;
}
```

Lo más interesante de este código es la animación que hacemos por medio de TweenMax: el botón ya fue visitado; entonces, le indicamos el color que llevan los elementos del sitio que fueron visitados.

Luego, igualamos **lastId** con el id de sección actual (**actualId**).

# Recordar el estado de la navegación

Manejando unas pocas variables, podemos indicar de manera sencilla qué sección fue visitada, cuál no lo fue y en qué sección se encuentra el usuario. Ahora bien, estos recursos no alcanzan cuando queremos lograr persistencia en esta información. Por suerte, Flash cuenta con un potente recurso mediante el cual podemos almacenar información de una película en la computadora del usuario: los **SharedObjects**.

## Almacenar información en la máquina del usuario

La clase **SharedObject** nos permite almacenar y recuperar información de una computadora o de un servidor. En nuestro caso, nos interesa guardar el historial de navegación en la máquina del usuario. La función de la clase **SharedObject** es muy similar a la de las *cookies* de los navegadores. Recordar al usuario las secciones visitadas es un modo sencillo se simplificar su visita considerando sus acciones anteriores. En **www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager03.html** especificamos el espacio para guardar datos.

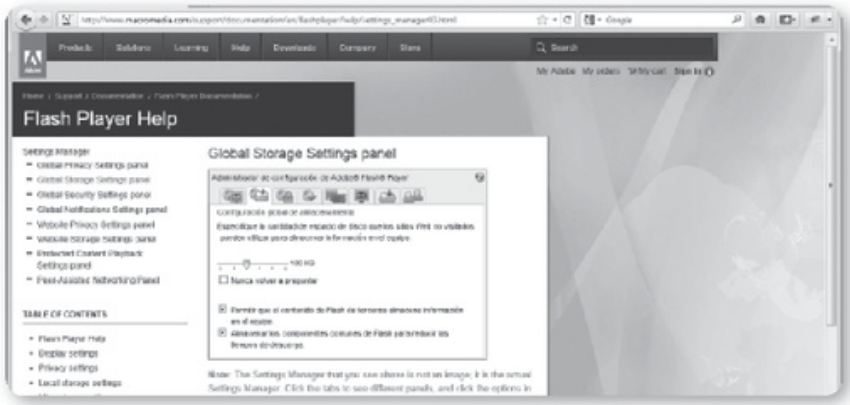



**Figura 7.** En este sitio es posible especificar la cantidad de espacio del disco para datos.

## Creación de SharedObjects

Por medio del método **getLocal()**, indicamos el nombre del objeto por crear, y usando la propiedad **data**, indicamos la información por almacenar. Si bien este recurso es de utilidad, no debemos abusar de esta posibilidad, y saber qué información resulta útil recordar y cuál no. De esta sencilla manera, creamos un nuevo **SharedObject** (instancia **SO**) y definimos la variable **siteName**, a la cual igualamos con el nombre del sitio que obtenemos del modelo, es importante que tengamos en cuenta que esto lo logramos simplemente implementando las tres sencillas líneas de código que mostramos a continuación:

```
var siteTile:String = "myFrameworkSite"
SO = SharedObject.getLocal(siteTile, "/");
SO.data.siteName = FrameworkModel.getInstance().siteTitle;
```

De todos modos, lo que realmente nos interesa es almacenar la navegación. En primer lugar, preguntamos si ya existe un historial almacenado. De no existir, lo creamos:

```
if (SO.data.history == undefined){ SO.data.history = new Object(); }
```

Dentro del objeto **history** que almacenamos mediante la propiedad **data** de la clase **SharedObject**, vamos a guardar la información de las secciones que fueron visitadas. Esto se hace dentro de la función **setActive()**:

```
SO.data.history[actualId] = actualId;
var □ush:String = SO.□ush();
```

### REPRODUCTOR DE VIDEO DE YOUTUBE

Al comienzo del capítulo dijimos que se puede dotar de recursos de usabilidad prácticamente a cualquier elemento multimedia. El reproductor de videos de **YouTube** es un buen ejemplo de esto: por medio del teclado, podemos controlar la totalidad de la reproducción de un video, su volumen y sus opciones.

Cada vez que ingresemos a la función, creamos una nueva propiedad para el objeto **history**, la cual llevará por nombre el id de sección actual (**actualId**), y le asignamos su mismo valor (**actualId**). De esta manera, estaremos seguros de que se irán almacenando las secciones que fueron visitadas por el usuario dentro de nuestro sitio web.

Por medio del método **flush()**, le indicamos a FlashPlayer que almacene esta información en la computadora del usuario.

Con estos sencillos pasos, basta para saber qué sección del sitio se visitó. Para representar esto visualmente, lo hacemos desde el bucle **for** con el que generamos el menú:

```
(SO.data.history[i] == i) ? buttonContainer.visited = true : buttonContainer.
    visited = false;
(SO.data.history[i] == i) ? buttonTextField.textColor = VISITED :
    buttonTextField.textColor = UNVISITED;
```

Si existe una propiedad llamada **i** dentro del objeto historial, y esta es igual al ciclo actual del **for**, sabemos que el elemento que se está creando ya fue visitado anteriormente por el usuario. En ese caso, modificamos la propiedad **visited** del **movieClip buttonContainer** y asignamos el color que corresponda al campo de texto: **VISITED** de haber sido visitado, o **UNVISITED** en el caso contrario.


**Figura 8.** Al ingresar en otra oportunidad, la película identificará las secciones que han sido visitadas con un color distinto.

# Navegación accesible

Del mismo modo en que aplicamos criterios de usabilidad a nuestro menú, podemos considerar algunas ventajas que brinda la accesibilidad e implementarlas de una forma muy sencilla.

## Indicar un orden de tabulado

No todos los usuarios tienen la posibilidad o la capacidad de navegar el contenido a través de un *mouse*. Con dos sencillas líneas, podemos hacer que el menú sea navegable por medio de la tecla **TAB**. Usando la propiedad **tabEnabled**, hacemos que un objeto **MovieClip** sea considerado en la lista de elementos al tabular, y la propiedad **tabIndex** indica la posición en el orden del tabulado.

Cuando le asignamos la variable **i**, hacemos que el menú se navegue con la tabulación en el orden en el cual se creó, tal como vemos a continuación:

> ES IMPORTANTE PERMITIR QUE EL MENÚ SEA NAVEGABLE A TRAVÉS DE TAB

```
buttonContainer = new MovieClip();
buttonContainer.tabEnabled = true;
buttonContainer.tabIndex = i;
```

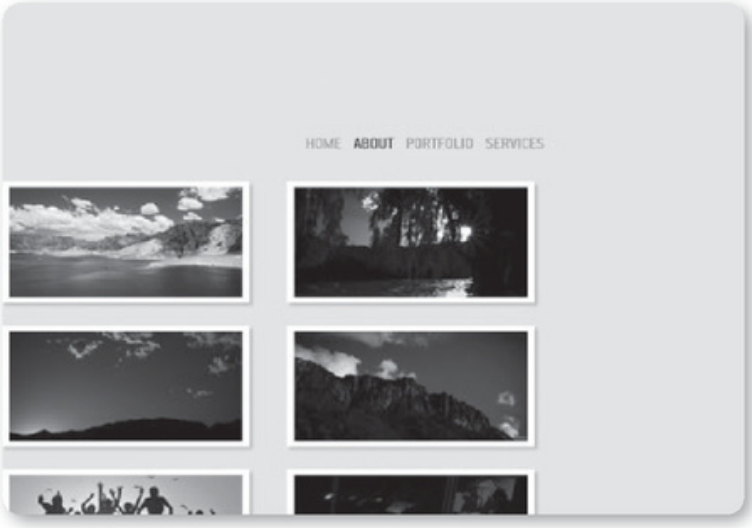



**Figura 9.** Al presionar **TAB**, un recuadro amarillo indica el foco. Con **BARRA ESPACIADORA** o **ENTER**, accedemos a la sección.

# Generar contenido para screen readers

Es necesario recordar que uno de los atributos que definimos para el nodo **site** del XML lleva por nombre **accessibility**. De esta forma, antes de aplicar soluciones de accesibilidad, debemos preguntar por esta variable. Esta información se toma del modelo:

```
hasAccessibility = FrameworkModel.getInstance().siteAccessibility;
```

## Implementación de la clase AccessibilityProperties

De haber definido el valor del atributo **accessibility** en **true**, hacemos uso de la clase **AccessibilityProperties**:

```
if(hasAccessibility){
    buttonContainer.accessibilityProperties = new AccessibilityProperties();
    buttonContainer.accessibilityProperties.name = vector[i].id;
    buttonContainer.accessibilityProperties.shortcut = "Ctrl+" + i.toString();
    buttonContainer.accessibilityProperties.description = "desde aqui puede
        acceder a la seccion " + vector[i].id + " del sitio";
}
```

Esta clase nos permite crear información alternativa dentro de nuestra película para que sea tenida en cuenta por *software* adecuado para la usabilidad, como puede ser un lector de pantalla (*screen reader*).

- Utilizamos la propiedad **name** para indicarle el nombre del objeto.
- Utilizamos la propiedad **shortcut** para que el lector de pantalla sepa cuál es el atajo a esa sección en caso de que existan algunos (lo crearemos próximamente).
- También tengamos en cuenta que utilizamos el campo **description** para asignar una descripción para el objeto.

En general, no se toma en consideración este tipo de información en el desarrollo, pero no hay que perder de vista que, con estas cuatro líneas de código, podemos brindarles una inmensa solución a los usuarios que tengan algún tipo de incapacidad.

## Uso de la clase Accessibility

Para que la implementación que llevamos a cabo surta efecto, en primer lugar, preguntamos si definimos criterios de accesibilidad para nuestro sitio. De ser así, por medio de la propiedad **active** averiguamos si hay activo algún lector de pantalla y si la aplicación se está comunicando con él. Finalmente, de darse estas condiciones, le indicamos a Flash Player que aplique los cambios realizados al emplear la clase **AccessibilityProperties**. Esto se efectúa por medio del método **updateProperties()** de la clase **Accessibility**:

```
if(hasAccessibility) if (Accessibility.active) { Accessibility.updateProperties();
```


**Figura 10.** JAWS es un lector de pantalla gratuito que podemos descargar desde el sitio **www.freedomscientific.com**.

# Forma alternativa de navegación I: shortcuts de teclado

Es importante tener en cuenta que asignar un valor a la propiedad **shortcut** de la clase **AccessibilityProperties** no crea el atajo de teclado a dicha sección. Simplemente, es útil indicarlo para que el lector de pantalla le informe al usuario cuál es el atajo que puede tomar a dicha sección. Veamos una forma sencilla para utilizar el teclado con el fin de navegar entre las secciones del sitio y facilitar aún más, de esta manera, la navegación mientras alguien se encuentra de visita en nuestro sitio web.

## Clase FrameworkAccessibility

Desde **FrameworkMain**, iniciamos la clase **FrameworkAccessibility**, pasándole como parámetro al método **init()** una instancia del escenario:

```
if (framework.getAccessibility()) FrameworkAccessibility.init(stage);
```

Tengamos en cuenta que esta clase nos permitirá navegar el sitio sin que sea necesario hacer uso del *mouse*. Dentro del método **init()**, agregamos una serie de valores al vector **keysVector** y asignamos un *listener* al escenario para detectar cuando se están presionando teclas. Los valores que añadimos por medio del método **push()** al vector **keysVector** indican el **keyCode** (código de tecla) que corresponde a las teclas de los números del teclado: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0:

```
public static function init(stageReference:Stage)
{
    keysVector.push(49, 50, 51, 52, 53, 54, 55, 56, 57, 48);
stageReference.addEventListener(KeyboardEvent.KEY_DOWN, checkKeyboard,
    false, 0, true);
}
```

Al presionarse una tecla, llamamos al *handler* **checkKeyboard()**, dentro del cual validamos el *input* del teclado: en primer lugar, preguntamos si el valor de la propiedad **keyCode** es menor o igual que 48 o mayor o igual que 56. De esta manera, nos cercioramos de que la tecla que se presionó corresponde a un número:

### TABULADO EN FORMULARIOS DE CONTACTO

Debemos saber que las propiedades denominadas **tabEnabled** y **tabIndex** son muy útiles al momento de crear formularios de contacto. A su vez, Flash cuenta con eventos para detectar el foco sobre los campos de texto (**FocusEvent.FOCUS_IN** y **FocusEvent.FOCUS_OUT**). Estos ayudan a tener un control en tiempo real de los campos y de las validaciones que hagamos.

```
private static function checkKeyboard(e:KeyboardEvent):void
{
    if (e.keyCode >= 48 && e.keyCode <=57){
```

Luego, hacemos uso del método **getPositionByKeyCode()**. Esta función recibe como parámetro el **keyCode** correspondiente a la tecla que se presionó, y nos devuelve a qué posición corresponde:

```
private static function getPositionByKeyCode(keyCode:Number):Number {
    var i:uint;
    var total:uint = FrameworkModel.getInstance().siteMenuVector.length;
    for (i = 0; i < total; i++) {
        if (keysVector[i] == keyCode) {
            return i;
        }
    }
    return NaN;
}
```

Utilizamos esta función porque no tenemos otra forma de saber a qué tecla corresponde cada código. De esta manera, comparamos el **keyCode** con los elementos almacenados en el vector y obtenemos la posición: el **keyCode** 49 corresponde a la tecla 1, el 50 a la tecla 2, el 51 a la tecla 3, etcétera. Esta información se obtiene de la posición de cada valor dentro del vector. Finalmente, si la tecla presionada es un número y, a su vez, se está presionando la tecla **CTRL** (**e.ctrlKey**), obtenemos la información necesaria del modelo y hacemos uso del método **setActive()**

## PRUEBAS CON LECTORES DE PANTALLA

Es aconsejable descargar un lector de pantalla y poner a prueba el código que estamos creando. A su vez, es muy importante experimentar el sitio de la misma manera en que lo hace una persona con discapacidades visuales: debemos corroborar que pueda navegar por el contenido con los ojos cerrados.

del menú, pasándole la posición a la cual queremos navegar. A su vez, implementamos la clase **FrameworkSectionManager**, que se ocupa de toda la navegación del sitio (la veremos próximamente); a continuación, podemos ver un ejemplo del caso comentado:

```
if (!isNaN(getPositionByKeyCode(e.keyCode)) && e.ctrlKey){
    var id = FrameworkModel.getInstance().siteMenuVector[getPositionByKeyCo
    de(e.keyCode)].id;
    FrameworkSectionManager.getInstance().setValue(id);
    Menu.getInstance().setActive(getPositionByKeyCode(e.keyCode));
}
```

Con esta sencilla clase, basta para que el usuario pueda navegar el sitio sin necesidad de utilizar el mouse de la computadora.

# Forma alternativa de navegación II: menú contextual de Flash

La ventaja que nos brinda manejar toda la información desde el modelo (**FrameworkModel**) es que resulta sumamente sencillo implementar nuevas opciones para el sitio: suponiendo que se agreguen varias secciones más, **no precisaremos hacer absolutamente nada para que se genere el menú del sitio**, ni para que se asignen shortcuts de teclado a cada sección, y tampoco en el caso que veremos a continuación, en el que vamos a agregar las secciones del sitio al menú contextual de Flash.

## SHAREDOBJECT Y SHORTCUTS

Si bien empleamos la clase conocida como **SharedObject** para recordar el estado del menú de navegación, también podemos acceder a darle un sinfín de aplicaciones adicionales. Una posibilidad es que, en caso de detectar software de accesibilidad en la PC, recordemos esta información y le comuniquemos al usuario sobre las maneras alternativas de navegación.


**Figura 11.** Navegación desde el menú contextual de Flash.

De la misma manera y del mismo lugar en el que iniciamos la clase **FrameworkAccessibility**, también iniciamos la clase **FrameworkContextMenu**. La diferencia es que el método **init()** de la clase **FrameworkContextMenu** nos devuelve un menú contextual.

A continuación veremos de qué manera agregar las secciones de nuestro sitio: para tener otra manera de acceder a las secciones.

```
if (framework.getContextMenu()) contextMenu = FrameworkContextMenu.
    init(framework.getFullScreen());
```

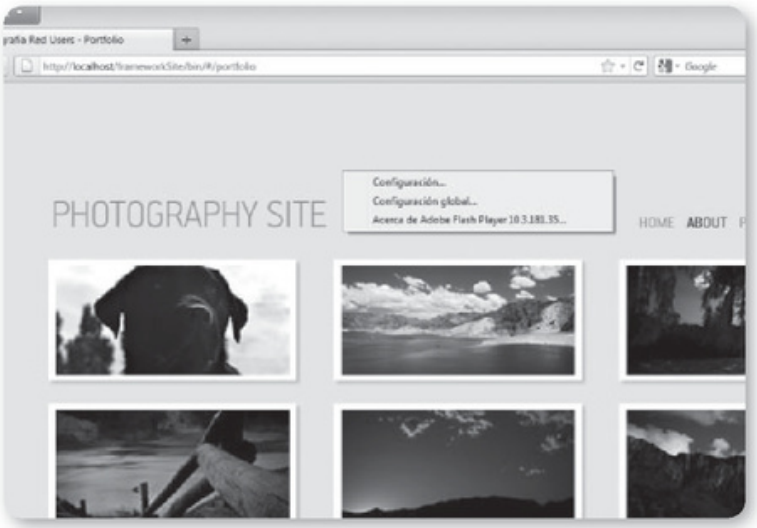



**Figura 12.** Es posible usar **`hideBuiltItems()`** para ocultar los elementos del menú que se incluyen por defecto.

```
public static function init(fullScreenMode:Boolean):ContextMenu
{
    menu.hideBuiltInItems();
    customItems = new Array();
    separator = true;
```

A continuación, agregamos al menú el título del sitio:

```
var title:String = FrameworkModel.getInstance().siteTitle;
var projectName:ContextMenuItem = new ContextMenuItem(title);
customItems.push(projectName);
```

Y luego llamamos a la función **addCustomMenuItems()**, pasándole por parámetro el vector que contiene toda la información del sitio:

```
addCustomMenuItems(FrameworkModel.getInstance().siteMenuVector);
```

Dentro de esta función **addCustomMenuItems()**, parseamos la información que contiene el array, y preguntamos por la propiedad **menu** de cada objeto. Recordemos que podíamos indicar desde el XML si queríamos que cada sección contara con menú contextual o no. De darse la condición, utilizamos el método **push()** del array **customItems** para almacenar dentro cada ítem del menú.

```
private static function addCustomMenuItems(pages:Vector.<Object>):void
{
    for (var i:int = 0; i < pages.length; i++)
    {
        if (pages[i].menu) customItems.push(createItem(pages[i]));
    }
}
```

La creación de cada ítem del menú contextual (**ContextMenuItem**) se realiza dentro de la función **createItem()**. Dentro de ella también

le indicamos que, al seleccionarse cada ítem, se llame a la función **onGoto()**, tal como podemos ver en el siguiente código:

```
private static function createItem(item:Object):ContextMenuItem {
    var cmi:ContextMenuItem = new ContextMenuItem(item.id);
    cmi.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, onGoto,
        false, 0, true);
    return cmi;
}
```

Finalmente, dentro de la función **init()** le indicamos al menú contextual, por medio de su propiedad **customItems**, que los ítem por utilizar están dentro del array en el cual los almacenamos:

```
menu.customItems = customItems;
```

Al detectarse el evento **MENU_ITEM_SELECT** y ejecutarse la función **onGoto()**, hacemos uso, una vez más, del menú (**Menu.as**), e indicamos la página que tiene que señalar como activa mediante el método **setActive()**. Una vez más, empleamos la clase **FrameworkSectionManager**, que veremos a continuación, para efectuar la carga y descarga de los contenidos:

```
private static function onGoto(event:ContextMenuEvent):void {
    var text:String = event.target.caption;
Menu.getInstance().setActive(FrameworkModel.getInstance().
    sitePageByUrlFriendly(event.target.caption).sid);
    FrameworkSectionManager.getInstance().setValue(event.target.caption)
}
```

## TAMAÑO DE CUERPO TIPOGRÁFICO

No olvidemos que cuando creamos contenido usable y accesible, debemos tener en cuenta el tamaño del cuerpo tipográfico. Por medio de las clases **TextFormat** y **TextField**, es sumamente sencillo brindarle al usuario la posibilidad de modificar el tamaño de los textos de nuestro sitio.

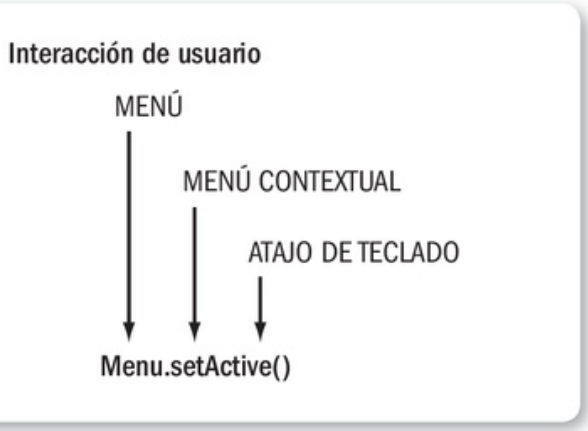



**Figura 13.** Desde el menú contextual o con atajos de teclado, llamamos a `setActive()` de la clase **Menu**.

# Navegación con deep linking: SWFAddress

La navegación es, sin duda, la parte más compleja de todo nuestro entorno de trabajo; no necesariamente por las tareas que se realizan, sino porque se ven involucrados varios procesos, dentro de los cuales cada clase tiene una función específica. En el capítulo anterior analizamos el modelo y la vista del sitio, y ahora es el turno del controlador. Si logramos entender con claridad sus procesos, indudablemente tendremos resuelta la lógica del funcionamiento de nuestro espacio de trabajo.

Al momento de desarrollar estas clases, pensamos en una solución definitiva al problema del deeplinking en Flash. Si bien esto es ventajoso, hay que dedicarle un tiempo considerable para comprender de qué manera manipulamos esta información.

## ASUAL.COM

El deeplinking es sin dudas una de las mejores implementaciones que le podemos hacer a un sitio a la hora de pensar en criterios de usabilidad para el mismo. SWFAddress fue desarrollada por Asual. Podemos encontrar más información en el sitio web **www.asual.com**: esta implementación también es funcional en Ajax.


**Figura 14.** Por medio de **SWFAddress**, empleamos la barra de navegación para indicar la sección del sitio a la cual queremos ir.

# Funcionamiento del controlador

- Debe cargar y descargar las secciones y el preloader. Para la carga de contenidos utilizamos la clase **FrameworkLoaderManager**.
- Luego, debe inicializar una clase que se encargue de indicar cuándo tiene que comenzar una carga. De esto se ocupa la clase **FrameworkSectionManager**.

Una carga se puede iniciar por iniciativa de un usuario, o bien al inicializarse el *framework*. Dependiendo de cuál de estas dos situaciones se den, el controlador operará de dos maneras distintas:

## Carga por primera vez de los contenidos

En este caso, el controlador realiza los siguientes pasos:

- Inicia la transición de entrada del preloader.

### TAMAÑO, CONTRASTE Y USO DE AUDIO

Flash brinda una gran cantidad de recursos para modificar el tamaño de nuestro contenido y el contraste de los elementos que lo componen. Por otro lado, es ideal para manejo de sonidos: puede ser útil emplear piezas de audio con información del sitio para aquellas personas que tengan dificultades visuales.

- Realiza la carga de la página y muestra la información en el preloader.
- Una vez concluida la carga de la página por mostrar, inicia la transición de salida del preloader.
- Cuando el preloader concluyó su salida, inicia la transición de entrada de la página por mostrar.

El siguiente gráfico seguramente ayude a aclarar estos pasos:

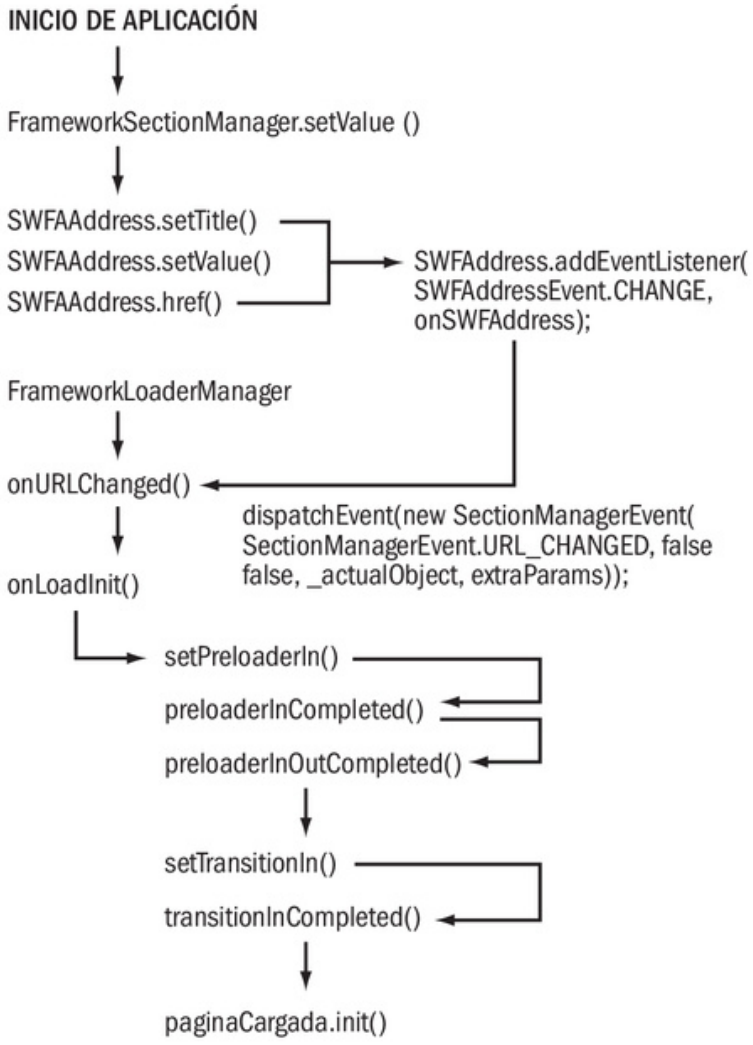



**Figura 15.** En este diagrama podemos ver la secuencia de clases, métodos y llamadas al ingresar en el sitio por primera vez.

## Carga por medio de interacción del usuario

A diferencia del caso anterior, en esta situación tenemos una página que descargar, por lo tanto debemos considerar lo siguiente:

- Inicia la transición de salida de la página actual.
- Una vez concluida la transición de salida de la página actual, inicia la transición de entrada del preloader.
- Concluida la transición de entrada del preloader, este comienza la carga de la próxima página por mostrar.
- Una vez cargada la próxima página, comienza la transición de salida.
- Concluida la transición de salida del *preloader*, empieza la transición de entrada de la nueva página.

Una vez que **FrameworkLoaderManager** carga de forma exitosa el *preloader* del sitio, se ejecuta el *handler* **preloaderReady()**:

```
public function FrameworkController(model):void {
    this.frameworkModel = model;
    FrameworkLoaderManager.getInstance().addEventListener(LoaderManagerE
        vent.LOAD_COMPLETED, preloaderReady, false, 0, true);
FrameworkLoaderManager.getInstance().loadContent(Framework.getInstance().
    getPreloader());
}
```

En esta instancia podemos iniciar la clase **FrameworkSectionManager**. Por un lado, le otorgamos el evento **SectionManagerEvent.URL_CHANGED**, asignándole como *handler* la función **onURLChanged()**, y por otro, iniciamos la clase mediante el método **init()**:

```
private function preloaderReady(event:Event):void {
    preloaderPage = FrameworkLoaderManager.getInstance().page;
    FrameworkLoaderManager.getInstance().removeEventListener(LoaderManag
        erEvent.LOAD_COMPLETED, preloaderReady);
    FrameworkSectionManager.getInstance().addEventListener(SectionManager
        Event.URL_CHANGED, onURLChanged);
    FrameworkSectionManager.getInstance().init();
}
```

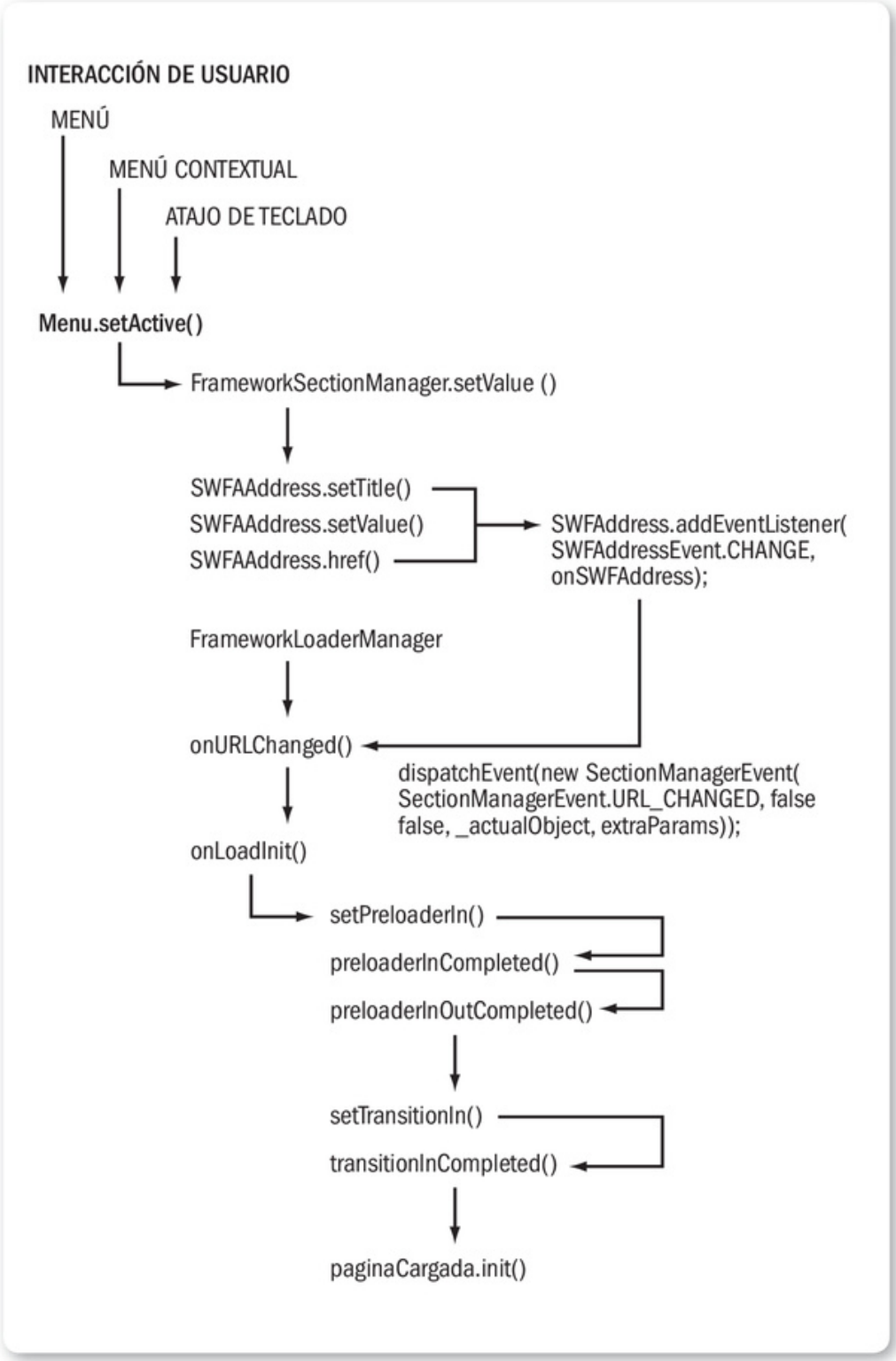



**Figura 16.** En esta imagen podemos ver la secuencia de eventos que corresponden según la interacción del usuario.

Desde la función **onURLChanged()** del controlador comienzan a desarrollarse las funciones que cargan y descargan el contenido de nuestro sitio. Abarcaremos este tema en el último capítulo del libro. Por el momento, veamos el funcionamiento de **FrameworkSectionManager**. Al ejecutarse el método **init()** de dicha clase, inicializamos **SWFAddress**.

# Introducción a SWFAddress

**SWFAddress** es una pequeña y potente librería que nos permite utilizar el *deep linking* del navegador. Además de permitirnos el uso de los botones atrás y adelante del navegador, **nos brinda la posibilidad de acceder a una sección específica del sitio desde la barra del navegador y de recibir parámetros adicionales en la URL**. Iniciamos **SWFAddress** desde el método **init()** de **FrameworkSectionManager**:

```
public function init():void{
    SWFAddress.addEventListener(SWFAddressEvent.CHANGE,
        onSWFAddress);
    _menuVector = FrameworkModel.getInstance().siteMenuVector;
    _defaultPage = FrameworkModel.getInstance().defaultPage;
}
```

El siguiente *listener* se ocupará de ejecutar **onSWFAddress()** cada vez que se produzca algún evento. Tengamos en cuenta que este listener se encargará de detectar cada vez que se produzca algún evento dentro de la navegación. Cuando esto suceda, llamamos a **onSWFAddress()**.

```
SWFAddress.addEventListener(SWFAddressEvent.CHANGE, onSWFAddress);
```

## TESTEOS

Al trabajar con contenido usable y accesible, es imprescindible que realicemos testeos sobre la aplicación: el hecho de que terceros hagan uso de nuestros desarrollos nos brindará información factible sobre los errores y los aciertos del trabajo que estamos llevando a cabo.

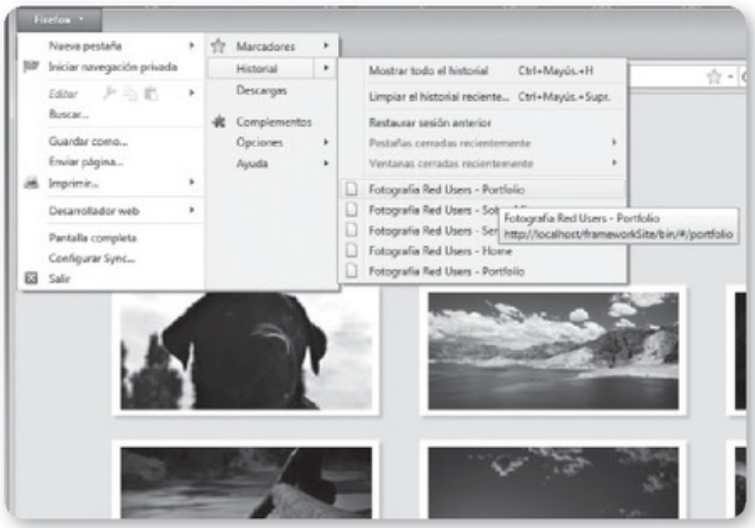



**Figura 17. SWFAddress** también nos permite ir almacenando el historial de navegación del sitio web.

## Cambiar de página con SWFAddress

Cada vez que queramos producir un cambio en la URL, llamamos a la función denominada **setValue()** de la clase. **Esto se hace desde el menú, desde el menú contextual y desde los atajos de teclado**. Lo importante es que, sea desde donde sea, le pasemos el id de la sección que queremos cargar, tal como vemos a continuación:

```
FrameworkSectionManager.getInstance().setValue(e.target.id);
```

La función **setValue()** recibe el id de la sección (**$idSection**) y los parámetros adicionales si es que se le pasa alguno:

```
public function setValue($idSection:String, $extraParams:String = null):void {
```

### VOLUMEN EN YOUTUBE

Le proponemos realizar una prueba: en primer lugar ingrese a YouTube, modifique el volumen de un video, cierre la ventana y luego vuelva a entrar. Podrá darse cuenta que el slider de volumen respeta la última posición que le asignó ¿Se imagina cómo fue hecho, verdad? En este capítulo se encuentra la respuesta.

A continuación, valida que la información sea correcta (**checkId()**), y en caso de ser así, obtiene la información del objeto:

```
var idObject:Object = checkId($idSection);
var urlFriendly:String;
var fileName:String;
var isSwf:Boolean = false;
if(idObject != null){
    urlFriendly = idObject.urlFriendly;
    fileName = idObject.src;
    if(ExtensionManager.getExtension(fileName) == "swf") isSwf = true;
}
```

**Figura 18.** En el último capítulo de este libro implementaremos una galería de imágenes, que recibe la ruta desde la URL del navegador.

## Abrir nueva página: SWFAddress.setValue()

Se obtiene la extensión del archivo por cargar (por medio de **EXtensionManager**); de ser **.SWF**, llama a **setValue()** de la clase **SWFAddress**:

```
if(isSwf){
    var strPath:String = urlFriendly;
    if($extraParams) strPath = strPath + "/" + $extraParams;
    SWFAddress.setValue(strPath);
    _currentSectionTitle = idObject.title;
```


**Figura 19.** Si bien el título del sitio no forma parte del menú de navegación, debe llevarnos a la home correspondiente.

## JERARQUÍAS VISUALES

A la hora de desarrollar cualquier tipo de aplicación, no debemos perder de vista la importancia de las jerarquías visuales: en el caso de un sitio web, el título suele ser más relevante que la botonera, y a su vez, esta suele estar en un sitio que nos garantiza una fácil interacción. A la hora de pensar en criterios de usabilidad y accesibilidad, es imprescindible tener un fluido feedback con los diseñadores de la aplicación para considerar estas jerarquías. Solo de esta forma lograrmos excelente resultados.

## Modificar el título de la barra del navegador: SWFAddress.setTitle()

Para modificar el título del sitio utilizamos el método **setTitle()**. Definimos el título concatenando lo siguiente:

- El nombre del sitio: **FrameworkModel.getInstance().siteTitle**.
- El delimitador para separar el nombre del sitio, del título de sección: **FrameworkModel.getInstance().siteDelimiter**.
- El título de la sección: **currentSectionTitle**.

```
SWFAddress.setTitle(FrameworkModel.getInstance().siteTitle + " " +
    FrameworkModel.getInstance().siteDelimiter + " " + _currentSectionTitle);
```

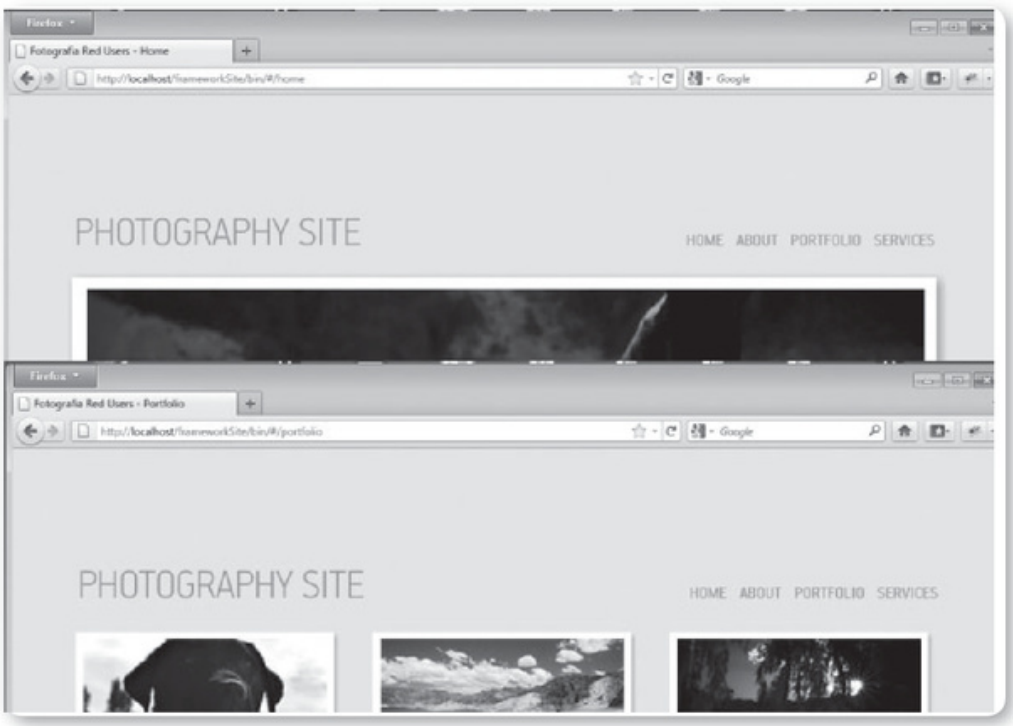



**Figura 20.** Al cambiar de sección, modificamos no solamente la URL del navegador, sino también el título del sitio.

## Abrir nueva página en blanco: SWFAddress.href()

En caso de que **ExtensionManager** indique que no se trata de un archivo **.SWF**, abrimos la nueva URL en una nueva página (seguramente se tratará de un archivo PHP o HTML) por medio del método **href()** de la clase:

```
    }else{
        SWFAddress.href(idObject.src,"_blank");
    }
```

Cualquiera sea el método que se ejecute desde **setValue()**, se ejecutará el *handler* **onSWFAddres()**, dentro del cual se llevan a cabo una serie de validaciones para saber si la URL contiene parámetros adicionales y si se produjo un cambio de sección. En caso de que haya un cambio, se ejecuta el evento:

```
SectionManagerEvent(SectionManagerEvent.URL_CHANGED, false, false,
    _actualObject, extraParams));
```

De lo contrario, si cambian los parámetros adicionales de la URL y no la sección, se ejecuta el evento:

```
dispatchEvent(new Event(FrameworkSectionManager.EXTRA_PARAMS_
    CHANGED));
```

Debemos tener en cuenta que estos eventos se ejecutan dentro del controlador del sitio (**FrameworkController**).

## Método onSWFAddress()

```
private function onSWFAddress(e : SWFAddressEvent) : void {
    e.stopPropagation();
    var strValue:String = e.value;
    _arrNewNavPath = strValue.split("/");
    _arrNewNavPath.shift();
    _arrParams = [];
    validatePath();
    FrameworkModel.getInstance().actualPage = _actualObject;
   var extraParams:String = _arrParams.join("/");
```

```
        if (_lastObject == _actualObject) {
            if(extraParams.length > 1){
                _extraParams = extraParams;
                dispatchEvent(new Event(FrameworkSectionManager.EXTRA_
                    PARAMS_CHANGED));
            }else {
                _extraParams = null;
            }
        }else {
            _extraParams = extraParams;
            _lastObject = _actualObject;
            dispatchEvent(new SectionManagerEvent(SectionManagerEvent.URL_
                CHANGED, false, false, _actualObject, extraParams));
        }
    }
```

# Ventajas de SWFAddress para la usabilidad del sitio

Trabajando las secciones del sitio de esta manera, logramos:

- Contar con la posibilidad de utilizar el historial del navegador.
- Contar con la posibilidad de utilizar los botones del navegador.
- Cargar el sitio en la sección que queramos y llegar directamente a ella, sin necesidad de pasar antes por la home.
- Que el usuario tenga la posibilidad de navegar o guardar una página específica dentro de nuestro sitio sin necesidad de acceder a la home.

## RESUMEN

Existen dos razones principales por las cuales no suelen implementarse soluciones usables y accesibles: en general, no se cuenta con los tiempos necesarios para hacerlo y hay muy poca información al respecto. En este capítulo proponemos una solución dual: por un lado, brindar información sobre lo que implican usabilidad y accesibilidad; y por otro, relegar el funcionamiento de soluciones usables y accesibles a clases que solamente hay que reutilizar, de forma que no sea necesario reprogramar su contenido.

# Actividades

## TEST DE AUTOEVALUACIÓN

1. ¿Qué es la **usabilidad**?
2. ¿Qué es la **accesibilidad**? ¿Qué tienen ambas en común y en qué se diferencian?
3. ¿Cuáles son las ventajas de hacer un sitio **accesible**? ¿Cuáles las ventajas de un sitio **usable**?
4. ¿En qué beneficia a la accesibilidad lo visto en el **Capítulo 4** de este libro?
5. ¿Para qué sirve la clase **SharedObject**? Además del uso que hicimos de esta clase en este capítulo, ¿que otros usos se le ocurren?
6. ¿Qué es un **screen reader** o lector de pantalla?
7. ¿Cuál es la ventaja de diferenciar los distintos estados de un menú de navegación?
8. ¿Para qué sirve la clase **Accessibility** de Flash?
9. ¿Para qué sirve la clase **AccessibilityProperties**? ¿Para qué se la usó en este capítulo?
10. ¿Para qué sirven las propiedades **tabEnabled** y **tabIndex**?

## ACTIVIDADES PRÁCTICAS

1. Piense y diseñe algún sistema de navegación.
2. Agréguele todas las funcionalidades vistas en este libro para que sea más usable.
3. Piense en algún otro uso interesante que se pueda hacer de la clase **SharedObject** e implemente la solución.
4. Haga que la información sea accesible para un lector de pantalla.
5. Utilice el teclado para generar algún sistema de navegación alternativo al analizado en el capítulo. Una opción puede ser emplear las teclas de dirección.

# *07 –

# Desarrollo en Flash para dispositivos

Una de las grandes ventajas que brinda Flash es la de poder pensar nuestros desarrollos para las más diversas plataformas: si bien la Web seguirá siendo un entorno apto para Flash Player, cada vez son más las posibilidades que nos brindan los dispositivos, y cada vez es más importante que pensemos en ellos a la hora de escribir nuestro código.

# Ideas preliminares

En general, a la hora de pensar en desarrollo para dispositivos, se entremezclan una serie de conceptos que debemos dejar en claro antes de abocarnos de lleno a este tema. En primer lugar, debemos entender a qué nos referimos con desarrollo para dispositivos: suele vincularse este término a los *smartphones*, pero, en verdad, la gama de productos que asoma detrás del concepto de dispositivo es mucho más amplia: las tabletas forman parte de este mundo; un GPS y una TV también: **cualquier dispositivo capaz de correr Flash Player 10.1 será considerado una plataforma apta para el desarrollo en Flash**.

**Figura 1.** Cada vez es mayor la cantidad de dispositivos que permiten ejecutar Flash Player.

# El comienzo de todo: Flash Player 10.1

Flash Player 10.1 marcó el comienzo de una nueva era en el desarrollo utilizando la Plataforma Adobe Flash: desde la llegada de esta versión, se marca un antes y un después, extendiendo las posibilidades del *plugin*: se puede exportar contenido hacia otro tipo de dispositivo que no sea necesariamente un sitio web, y se puede crear contenido de otra índole.

Esto es, sin dudas, la máxima ventaja que nos proporciona la Plataforma Adobe Flash: conociendo un único entorno y un único lenguaje de programación, podemos crear contenido de distinto tipo para diferentes soportes. Ahora bien, así como resulta ventajoso, no

debemos perder de vista uno de los conceptos más importantes que recalcamos en reiteradas ocasiones: el entorno define en gran medida el uso que hacemos de nuestras aplicaciones. No es lo mismo crear contenido web para ser visualizado en una computadora, que para ser visto en un *smartphone*; tampoco es igual la cantidad de memoria disponible en los distintos soportes, ni se cuenta con igual autonomía. Ni tampoco es igual la manera en la que se vive la experiencia ante los distintos soportes: desde una computadora, seguramente navegaremos un sitio web con determinada comodidad, al igual que una aplicación corriendo en una TV, mientras que una aplicación para un dispositivo móvil tal vez nos exija otro tipo de interacción y de experiencia, porque podemos estar haciendo uso de ella mientras caminamos o realizamos algún otro tipo de actividad. Debemos tener presentes estas cuestiones a la hora de comenzar un desarrollo.


**Figura 2.** Desde la versión 2.1, **Android** hizo viable la reproducción de contenido Flash en varios de los dispositivos móviles.

# ¿Hacia dónde va Flash? Open Screen Project

Una de las preguntas que suelen hacerse muchos desarrolladores es hacia dónde va Flash, en especial, desde la llegada de HTML5. En los primeros capítulos de este libro, nombramos la diferencia entre ambos soportes y su alcance. Flash no es un reemplazo de HTML, ni HTML es un reemplazo de Flash. Son dos tecnologías complementarias, y Flash

tiene la obligación de encontrarse un paso adelante de lo que se ofrece como estándar, principalmente, en lo que hace al desarrollo de videojuegos, de video y de RIAs. La mejor manera de comprender el enfoque que busca Adobe para Flash Player es entendiendo lo que motiva la creación y desarrollo del el proyecto **Open Screen Project**.

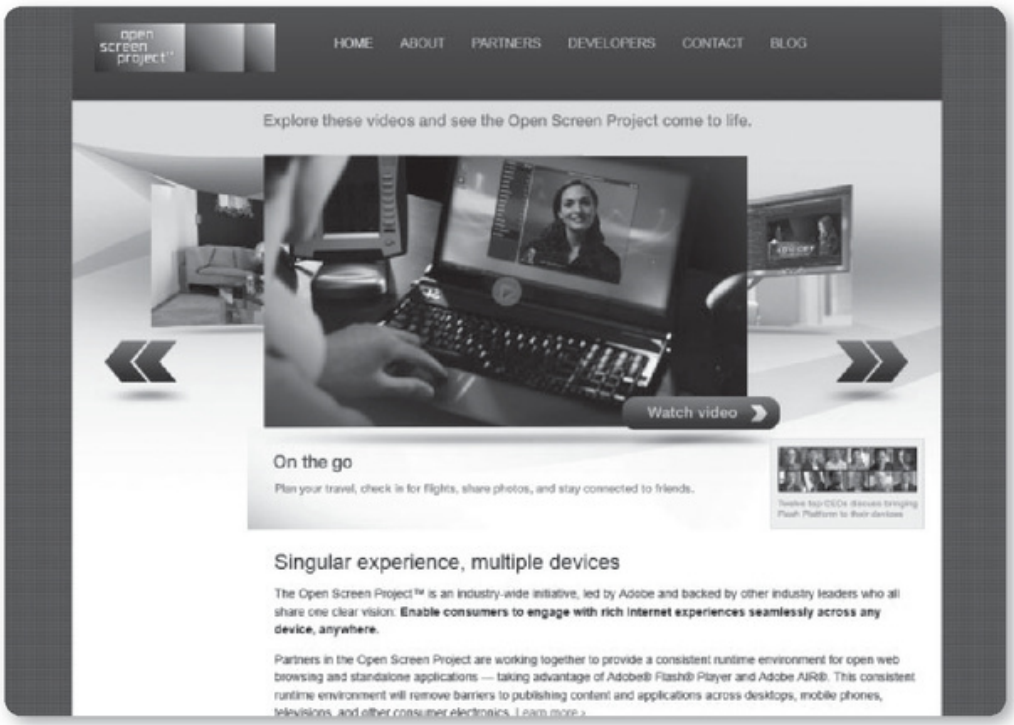



**Figura 3.** Se puede encontrar más información de Open Screen Project en su sitio web: **www.openscreenproject.org**.

*Open Screen Project* es un proyecto liderado por Adobe mediante el cual se busca brindarles a los usuarios la posibilidad de vivir experiencias ricas en Internet independientemente del dispositivo que

## OPEN SCREEN PROJECT

Entre las compañías que integran el proyecto **Open Screen Project** se encuentran **AMB**, **Cisco**, **Google**, **HTC**, **Intel**, **LG**, **Motorola**, **Nokia**, **NVIDIA**, **Palm**, **Samsung**, **Sony**, **Symbian** y **Toshiba**. Los avances de este proyecto están en clara evidencia.

se emplee para hacerlo y del lugar en el que uno se encuentre. Para lograrlo, Adobe trabaja en conjunto con más de 200 empresas del rubro informático: desde firmas líderes orientadas a la Web, hasta empresas clave en desarrollo de procesadores o de contenidos.

En este sentido, basta con tener en cuenta que **google**, **nvidia**, **Samsung**, **Sony**, **AMD**, **Cisco** e **intel**, entre tantas otras grandes empresas se encuentran detrás de este proyecto, para conocer y hacernos una idea de la magnitud y seriedad del mismo.

# ¿Cómo desarrollar para distintos dispositivos?

Este es el punto más fuerte que trataremos en este capítulo: ¿qué hacemos cuando tenemos en nuestras manos un desarrollo que debe visualizarse en distintos dispositivos, con diferentes prestaciones y diversos tamaños de pantalla?

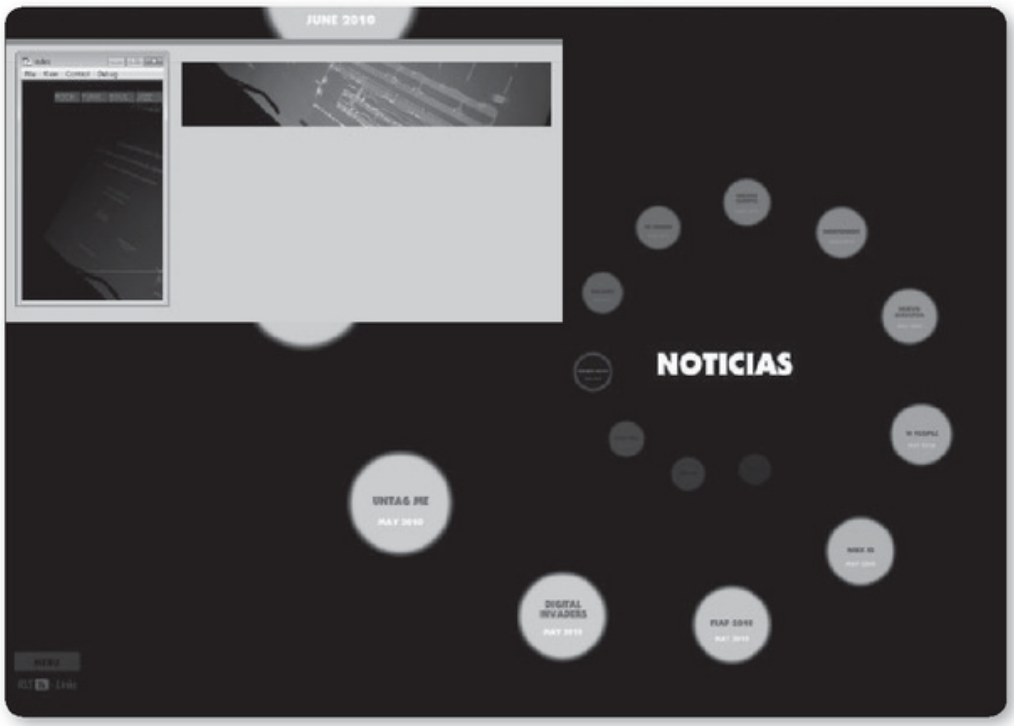



**Figura 4.** Grupo W presenta un sitio íntegramente desarrollado en Flash para ver desde una PC, y una alternativa para un smartphone.

## Generar distintas aplicaciones para distintos dispositivos

Esta es, sin dudas, la mejor alternativa. Ahora bien, hay una realidad detrás de esto a la cual no podemos escapar: lo que determina si tenemos posibilidades de generar distintas experiencias son los tiempos y el dinero con el que contemos. Lamentablemente, no siempre estas dos variables están a nuestro favor, y en ocasiones se hace prohibitivo pensar en diferentes implementaciones.

## Emplear condicionales para modificar el comportamiento de la aplicación dependiendo de su entorno

Podemos modificar el comportamiento de nuestras aplicaciones empleando condicionales. Por ejemplo, en el **Capítulo 4** de este libro vimos de qué manera distinguir cuando un usuario ingresa desde un *smartphone* y cuando lo hace desde una computadora: teniendo esta información dentro de la película, podemos tomar decisiones para mejorar la performance según desde dónde se visualice el contenido. Por ejemplo, podemos tener las fuentes en un archivo externo y, en caso de conectarse desde una computadora, las cargamos y empleamos, mientras que si la conexión se estableció desde un dispositivo móvil, utilizamos las fuentes de sistema.

> PODEMOS MODIFICAR EL COMPORTAMIENTO DE LAS APLICACIONES UTILIZANDO CONDICIONALES

Si bien esta solución puede ser útil en casos muy puntuales de desarrollos sencillos, es la menos recomendable de todas, ya que, al hacer esto, terminamos haciendo ilegible nuestro trabajo, y a mayor complejidad, mayor cantidad de condicionales y de validaciones necesitaremos. Muchas veces nos lleva más tiempo pensar en estas validaciones que implementar dos alternativas distintas.

## Redimensionar el contenido

Otra alternativa con la que contamos es trabajar con medidas relativas y lograr que el contenido se visualice de manera correcta independientemente del dispositivo que se esté empleando.

# Redimensionar el contenido para distintos tipos de dispositivos

De las tres técnicas anteriormente descriptas, nos interesa abarcar esta en particular. La ventaja que presenta sobre el resto es que no necesitamos hacer ningún tipo de validación, y podemos obtener buenos resultados sin importar el dispositivo en el cual se visualice el contenido y sin tener que encarar dos tipos de desarrollos distintos.

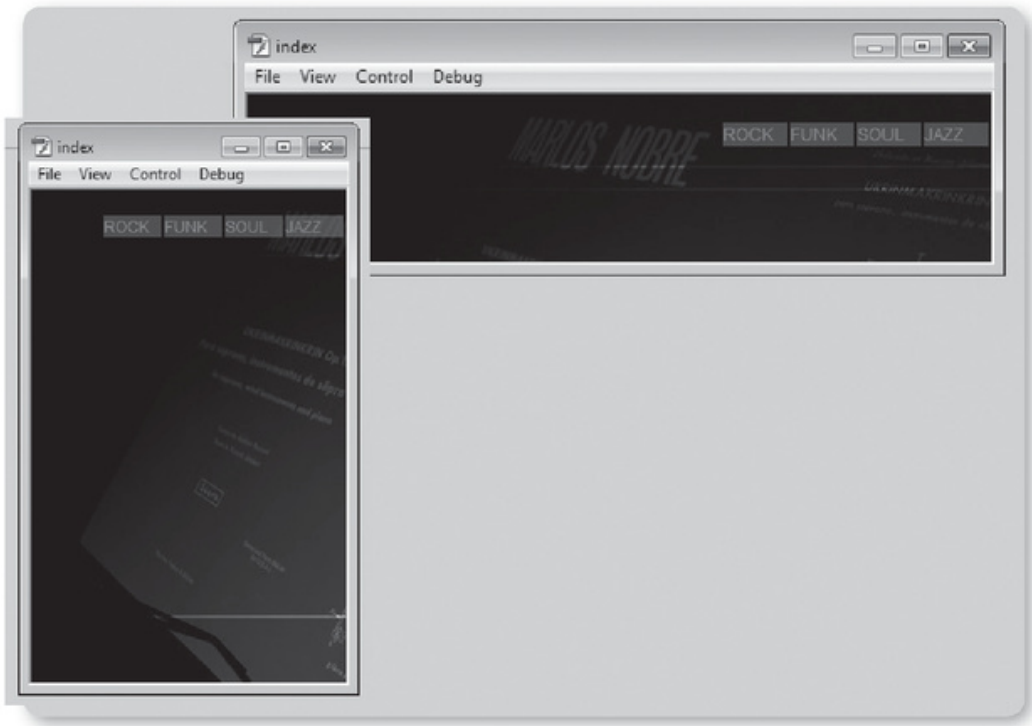



**Figura 5.** Ejemplo utilizando un `liquid layout`: accediendo a las propiedades del escenario, reacomodamos los elementos de la interfaz.

Recordemos lo que dijimos antes: si contamos con los tiempos y los recursos, la mejor opción siempre será generar distintas experiencias, y seguramente, HTML5 y CSS3 serán una mejor opción para crear contenido para dispositivos móviles que Flash, pero esta posibilidad no siempre está a nuestro alcance. Lo que sí podemos hacer es basarnos

en esta modalidad de desarrollo que es realmente útil y, de no contar con un *framework* generado específicamente para cada dispositivo, puede darnos grandes resultados escribiendo el código una única vez.

## Comenzar siempre por el escenario

Flash nos permite definir varias propiedades y eventos para el escenario (**Stage**). Al pensar en redimensionar el contenido, tenemos que indicarle a Flash que no escale la película, ya que de esa tarea nos ocuparemos nosotros. Esto se realiza por medio de la propiedad **scaleMode** del escenario, aplicándole como valor la constante **NO_SCALE** de la clase **StageScaleMode**:

```
stage.scaleMode = StageScaleMode.NO_SCALE;
```

A su vez, es necesario que indiquemos desde dónde se debe realizar la alineación del contenido. Para trabajar de esta manera, alineamos en el borde superior izquierdo de la pantalla. Esto se lleva a cabo por medio de la propiedad **TOP_LEFT** de la clase **stageAlign**:

```
stage.align = StageAlign.TOP_LEFT;
```

Una vez definidas estas propiedades, tenemos que averiguar cuándo se produce algún cambio en las dimensiones del escenario. Para esto, contamos con el evento **RESIZE**:

```
stage.addEventListener(Event.RESIZE, onResize, false, 0, true);
```

Al trabajar en entornos web, estamos acostumbrados a que este evento se ejecute cuando se modifica el tamaño de la ventana del navegador, pero el evento **RESIZE** actúa en otras dos oportunidades: cuando la aplicación se ejecuta por primera vez y cuando se cambia la orientación de un dispositivo móvil.

Debemos ser cuidadosos con esto y tomarlo siempre en consideración al desarrollar para smartphones.

## Emplear dimensiones relativas: stage.stageWidth y stage.stageHeight

Al redimensionar contenido para varios dispositivos, **stageHeight** y **stageWidth** nos acompañarán durante el desarrollo. **stageWidth** indica el tamaño del escenario en ancho, y **stageHeight**, la resolución en largo. Veamos cómo generar formas que se adapten al contenido:

```
mainContainer = new Sprite();
header = new Shape();
header.graphics.beginFill(0, 0.6);
header.graphics.drawRect(0, 0, stage.stageWidth, 50);
header.graphics.endFill();
footer = new Shape();
footer.graphics.beginFill(0, 0.6);
footer.graphics.drawRect(0, 0, stage.stageWidth, 50);
footer.graphics.endFill();
footer.y = stage.stageHeight - footer.height;
content = new Shape();
content.y = header.y + header.height + MARGIN;
content.graphics.beginFill(0, 0.6);
content.graphics.drawRect(0, 0, stage.stageWidth, Math.round(footer.y -
    content.y) - MARGIN);
content.graphics.endFill();
```


**Figura 6.** Las tres formas representan a **header**, **content** y **footer**. Observemos su posicionamiento.

De este modo, sabemos que las formas respetan las dimensiones del escenario, tanto en largo como en ancho. Ahora bien, ¿qué sucede si se redimensiona el navegador, o se cambia la orientación de un dispositivo móvil? Es aquí donde hacemos uso del evento **RESIZE** de Flash, a fin de detectar estos eventos y tomar una decisión que repercuta en nuestra interfaz. En el siguiente código podemos encontrar un claro ejemplo del evento que estamos comentando.

```
private function onResize(e:Event):void {
    buttonsContainer.y = 20;
    buttonsContainer.x = stage.stageWidth - buttonsContainer.width - MARGIN;
    footer.y = stage.stageHeight - footer.height;
    content.height = Math.round(footer.y - content.y) - MARGIN
    header.width = content.width = footer.width = stage.stageWidth;
}
```

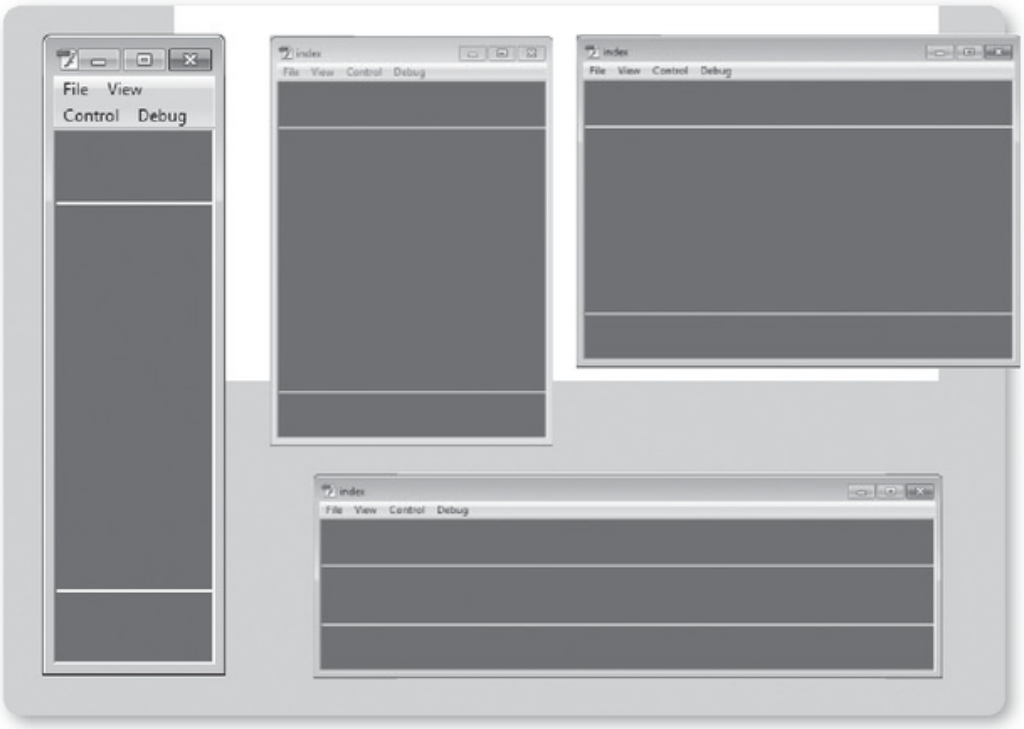



**Figura 7.** Al escalar el contenido o modificar la orientación, las formas se adaptan a las dimensiones de la pantalla.

# Uso de texto en dispositivos

El uso de campos de texto en dispositivos móviles merece ser tratado con especial cuidado: la forma de leer texto y de interactuar con los campos difiere considerablemente de la forma tradicional que nos propone una computadora.

## Campos de texto dinámicos: FTE

Pese a que su implementación lleve más trabajo y más líneas de código, es conveniente manipular el código por medio de **FTE** (*Flash Text Engine*). Este motor de texto fue incluido en la versión 10 de Flash Player y ofrece un mejor renderizado que la clase **TextField** de Flash. A su vez, el posicionamiento de los campos de texto es mucho más preciso. Veamos de qué manera generamos el menú de nuestro ejemplo:

```
var str:String = buttonsVector[i];
var fontDescription:FontDescription = new FontDescription("Arial","normal");
var format:ElementFormat = new ElementFormat(fontDescription, 12,
    0xff0066, 1);
var textElement:TextElement = new TextElement(str.toUpperCase(), format);
var textBlock:TextBlock = new TextBlock();
textBlock.content = textElement;
textBlock.baselineZero = TextBaseline.ROMAN;
var textLine:TextLine = textBlock.createTextLine(null, 300);
textLine.x = 0;
textLine.y = textLine.height+2;
button.addChild(textLine);
count+= Math.round(button.width + MARGIN);
buttonsContainer.addChild(button);
```

### REDIBUJO POR MEDIO DE ACTIONSCRIPT

Debemos tener en cuenta que para mostrar las áreas de redibujo por medio de ActionScript, empleamos el método **showRedrawRegions** del paquete **profiler**: **flash.profiler.showRedrawRegions(true, 0x0000FF)**. Al hacerlo de esta manera, podemos definir un color específico para las áreas.

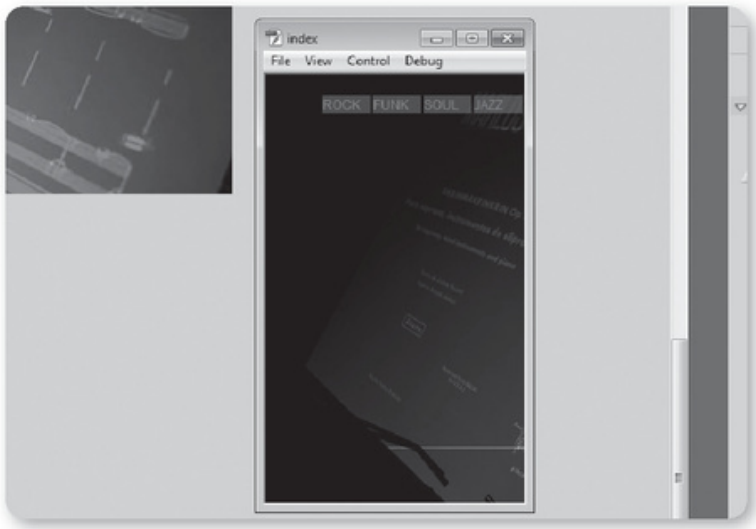



**Figura 8.** Campos de texto creados con clases de **FTE**. Para web móvil, es una mejor alternativa que **`TextField`** de Adobe Flash.

## Introducción de texto en dispositivos móviles

Cuando sea posible, evitemos usar campos de introducción de texto.

Todo aquel que cuente con un *smartphone* o algún otro tipo de dispositivo móvil sabe que la interacción con campos de texto no suele ser una buena experiencia. En la medida en que se pueda evitar su uso, vamos a brindar una interacción más fluida con la aplicación. En aquellos casos en los que realmente no nos quede otra alternativa que su implementación, una buena solución es desarrollar un algoritmo que nos permita autocompletar los campos o bien almacenar un historial con la información que ha sido suministrada por el usuario en otras oportunidades. En este libro hemos visto de qué manera hacer uso de **shared objects**, podemos darnos cuenta que llevar a cabo una solución de este tipo no debería demandarnos mucho tiempo, y podría serle de gran utilidad al usuario para quien implementamos el desarrollo.

### ALTERNATIVAS DE DESARROLLOS

Del mismo modo en que afirmamos que crear una aplicación para cada tipo de dispositivo es la mejor solución, también somos conscientes de que las posibilidades de desarrollo no siempre nos lo permiten. Muchas veces tenemos que lidiar con esto y emplear la solución que mejor se adapte a nuestra situación y al presupuesto con el que contamos.


**Figura 9.** Al buscar desde un smartphone, Google nos muestra los resultados en tiempo real y, a su vez, almacena el historial de consultas.

# Optimizaciones sobre el diseño de interfaz

Al trabajar con Flash, estamos acostumbrados a manipular elementos visuales, o bien trabajamos con diseñadores que se encargan del aspecto visual del sitio. Debemos tener en cuenta que en cualquiera de los dos casos, es importante recordar lo siguiente:

- Independientemente del tipo de desarrollo, mantener un diseño simple y una clara organización del layout nos permite hacer un mejor manejo de los contenidos.
- No debemos olvidar que quien está visualizando el contenido está usando un dispositivo, y no, una PC. La modalidad y las condiciones de visualización no son las mismas.

# Mostrar regiones de redibujo

Este recurso es útil ante cualquier tipo de desarrollo, pero nos será de especial ayuda al trabajar con dispositivos portátiles.

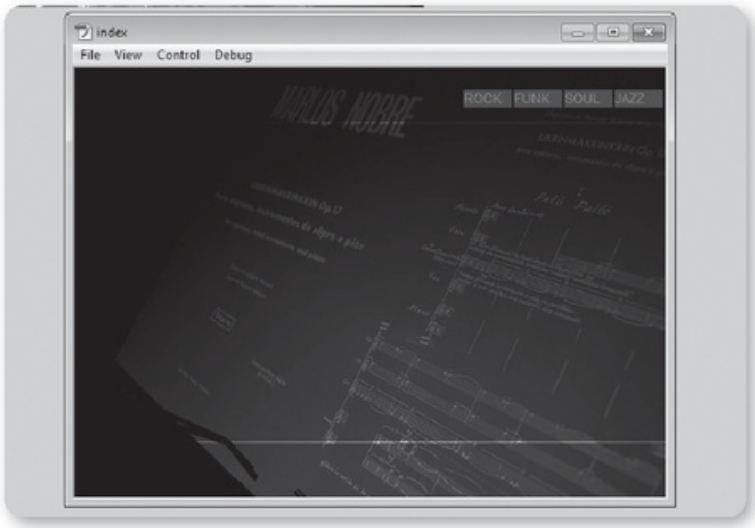



**Figura 10.** Podemos visualizar las regiones de redibujo accediendo desde el menú contextual de Flash.

Flash permite mostrar las áreas de redibujo de la película. Estas regiones muestran las áreas que Flash Player está procesando.

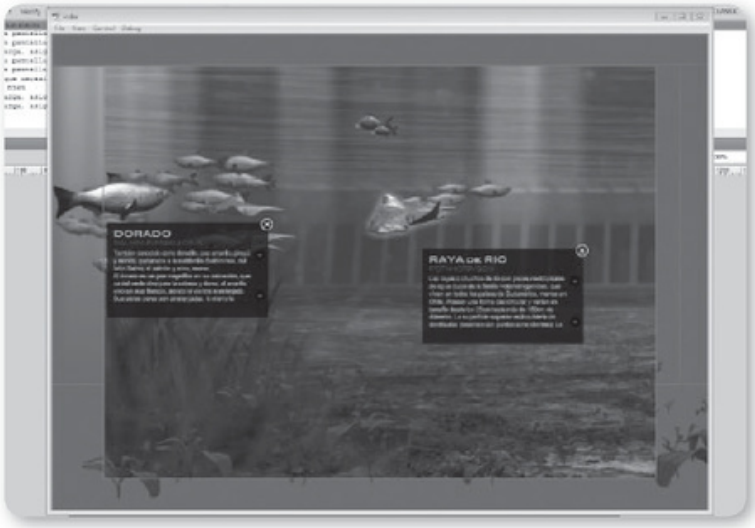



**Figura 11.** Visualización de las regiones de redibujo de una película Flash.

Cuando dejamos de usar un elemento de la lista de objetos, tenemos que eliminarlo con **removeChild()**. Si es conveniente mantenerlo en escena, habrá que emplear la propiedad **visible** igualada a **false**, y no la propiedad

**alpha** igualada a **0**. La diferencia entre las propiedades es que, cuando usamos **alpha**, el objeto se sigue procesando por más que no esté visible.

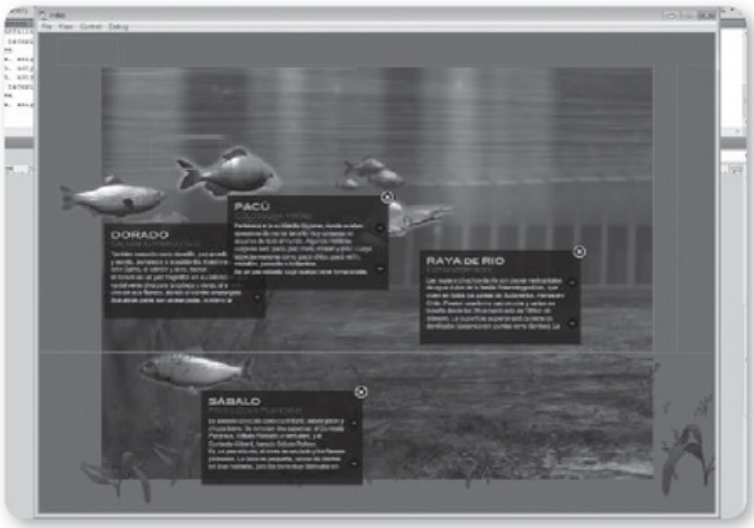



**Figura 12.** Aquí se ve con claridad un problema: gran actividad por medio de la región de redibujo sin ver ningún contenido.

Las regiones de redibujo ayudan para detectar **ENTER_FRAME** que no eliminamos a tiempo, o comportamientos que no tuvimos en cuenta.

# Aceleración por GPU

El uso de la GPU optimiza el proceso de renderizado de filtros, mapas de bits, videos, texto, animaciones en 3D y transformaciones de color.

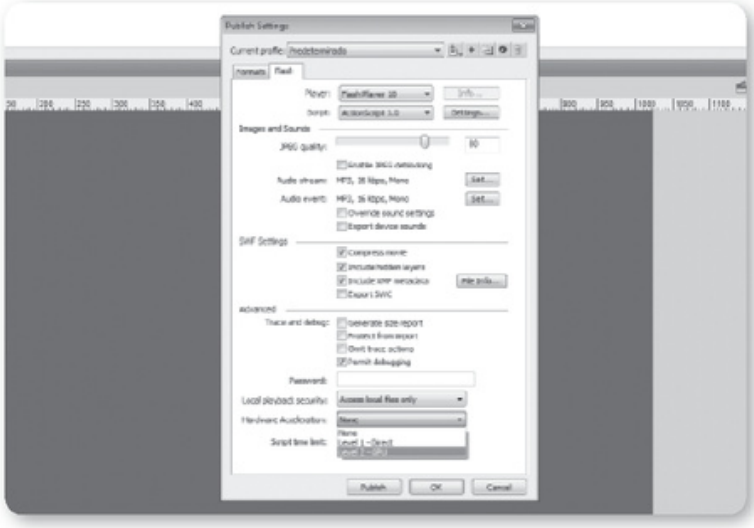


**Figura 13.** Para definir la opción, ingresamos enla **Configuración de Publicación** y seleccionamos el modo de aceleración adecuada.

# Optimizaciones sobre ActionScript

Este es el tipo de optimización que más nos interesa. Si bien las próximas páginas nos serán de ayuda, son simplemente una introducción al tema central del desarrollo para dispositivos: dedicaremos el próximo capítulo en su totalidad a todo lo referido a performance, uso de FPS y memoria disponible.

## Activación y desactivación de objetos

En reiteradas oportunidades a lo largo de este libro hemos usado el método **ADDED_TO_STAGE**, tanto para iniciar nuestro *framework* como para iniciar el comportamiento de una clase que extienda a un elemento visual (**Sprite** o **MovieClip**). Pero también contamos con el evento **REMOVED_FROM_STAGE**, el cual nos informa que se ha eliminado un elemento del escenario. Nos resulta útil emplear un *listener* para ese evento a fin de descargar de la memoria los elementos que hagan falta, y detener un sonido en caso de haber una reproducción, un video o lo que fuera.

GENERALMENTE USAMOS EL MÉTODO ADDED_TO_STAGE PARA EXTENDER A UN ELEMENTO VISUAL

De esta forma, es necesario que tengamos en cuenta que, al detectarse el evento **ADDED_TO_STAGE**, agregamos los objetos de nuestra película. Por el contrario, al detectarse el evento **REMOVED_FROM_STAGE**, los eliminamos.

### Clase Loader: emplear unloadAndStop() en vez de unload()

Utilizamos la clase **Loader** de ActionScript tanto para cargar como para descargar películas **.SWF** e imágenes (**.JPG**, **.PNG**). Al querer realizar una descarga, con anterioridad a la salida de Flash Player 10, empleábamos el método **unload()**. Si bien este procedía a realizar la descarga de la película, no se eliminaba ninguna referencia al contenido incluida en ella. Esto hizo que, durante un tiempo, hubiera que pensar en *hacks* para

evitar este *bug*. Una forma común de hacerlo era establecer conexiones entre las películas (clase **LocalConnection**) para remover absolutamente todas las referencias correspondientes antes de proceder a la descarga, entre otros métodos poco convencionales que, si bien funcionaron, distaban de ser la solución real al problema. En ese sentido, se introdujo la clase **unloadAndStop()** en la versión 10 de Flash Player.

El método **unloadAndStop()** realiza las siguientes tareas dentro de la película antes de proceder a su descarga:

- Detiene todos los MovieClips que contenga la película.
- Elimina cualquier instancia de la clase **Timer** que esté en ejecución.
- Detiene todos los sonidos que se estén ejecutando o cargando.
- Elimina cualquier evento que implique el uso de fotogramas (**ENTER_FRAME**, **EXIT_FRAME**, etc.).
- Elimina los *listeners* que se hayan asignado dentro de la película.
- Se encarga de realizar el cierre de todas las conexiones que hayan sido creadas mediante la clase **NetConnection**.
- Cierra cualquier conexión o reproducción que se haya iniciado, tanto de un video (clase **Video**) como de una cámara web (clase **Camera**) o de un micrófono (clase **Microphone**).
- Se encarga de detener cualquier tipo de conexión, carga o descarga (**sockets**, **fileReference**, etc.).

## RESUMEN

Si bien ActionScript 3.0 nos brinda todo su potencial a la hora de pensar en desarrollos para múltiples plataformas, no debemos perder de vista que el comportamiento de nuestras aplicaciones varía acorde a las prestaciones del dispositivo para el cual llevemos a cabo el trabajo. A su vez, cada dispositivo plantea una forma de interacción específica que desafía la manera convencional de pensar en nuestras aplicaciones: debemos comprender un dispositivo en profundidad antes de pensar en una aplicación para él. A continuación, dedicaremos un capítulo completo a la performance en Flash.

# Actividades

## TEST DE AUTOEVALUACIÓN

1. ¿Qué entendemos por dispositivo móvil?
2. ¿Qué es **Flash Lite**?
3. ¿Cuál es su diferencia con **Flash Player**?
4. ¿Qué es **Open Screen Project**? ¿Quiénes forman parte de él?
5. ¿Qué alternativas existen a la hora de desarrollar para distintos dispositivos?
6. ¿De qué manera debemos utilizar el texto dinámico?
7. ¿Por qué es preferible evitar el uso de campos de introducción de texto?
8. ¿Cómo se muestran las regiones de redibujo de Flash?
9. ¿Cuál es la diferencia entre los métodos **unload()** y **unloadAndStop()**?
10. ¿Con qué propiedades de la clase **Stage** obtenemos el largo y el ancho del escenario?

## ACTIVIDADES PRÁCTICAS

1. Desarrolle un sencillo diseño implementando lo incorporado en este capítulo y en el anterior.
2. Separe el contenido en no menos de tres secciones.
3. Genere un proceso de carga y descarga de los contenidos empleando el método **unloadAndStop()**.
4. Defina medidas mínimas y máximas para escalar el contenido.
5. Haga que la aplicación sea acelerada por medio de la GPU.

*08 –

# Optimización, memoria y FPS

La optimización de los recursos con los que contamos es una parte vital e imprescindible ante cualquier tipo de desarrollo que encaremos. En este capítulo veremos varios conceptos de ActionScript 3.0, técnicas e información que nos serán fundamentales para mejorar la performance de nuestras aplicaciones, optimizar el consumo de memoria y obtener un mejor rendimiento en las películas.

# Consideraciones previas

Si hay algo que hace de Flash una plataforma apasionante es su carácter multidisciplinario. Independientemente de los conocimientos que hagan falta, casi cualquier persona puede ponerse a jugar con Flash. Si bien nosotros lo utilizamos para emplear el potencial de AS3 y crear un *framework* orientado a la Web, el alcance de Flash es muy amplio: desarrolladores de videojuegos, animadores, artistas, diseñadores y publicistas, entre tantas otras personas, hacen uso de él. Lejos de considerar esto una falencia, a nuestro criterio, esta característica hace de Flash una gran herramienta. Ahora bien, a su vez, el hecho de que pueda ser empleado con tantos fines y por tantos usuarios con distintos perfiles hace que una película desarrollada en Flash pueda correr en las más diversas plataformas, como un sitio web, una aplicación para *smartphones*, una para una *tablet*, o bien como aplicación *stand alone* para una instalación multimedia, entre tantos otros casos. ¿Qué tiene que ver esto con la performance de una película? **Que, en gran medida, lo que define el buen o mal uso que hacemos de los recursos es el entorno dentro del cual ejecutamos una aplicación**. Unos pocos *kilobytes* en una aplicación que empleamos para una instalación corriendo en una computadora que tenga un procesador Core I7 no hará la diferencia, pero puede que sí la haga en una aplicación para un dispositivo móvil. Si bien podemos concedernos algunas libertades dependiendo de los recursos con los que contemos, siempre es una buena práctica hacer uso de aquellos de los que disponemos para garantizar una buena performance. Iremos descubriendo varios *tips* que nos permitirán ahorrar recursos y obtener un mejor rendimiento.

> SIEMPRE ES IMPORTANTE HACER LO POSIBLE PARA OBTENER UNA BUENA PERFORMANCE

## VELOCIDAD DE FPS

La decisión que se toma a la hora de definir los FPS de una película debe ser racional y estar sustentada en lo que el desarrollo nos exige: si no hay animación, no es necesario superar los 12 FPS. En caso de haberla, 24 FPS es suficiente para ver con fluidez los contenidos, y en general, no tiene sentido emplear más de 30.

# Tracker para FPS y memoria

A continuación, llevaremos a cabo el desarrollo de un *tracker* mediante el cual podamos obtener el rendimiento que está teniendo nuestra película dentro de determinada cantidad de muestras durante un período de tiempo.

**Figura 1.** La clase **SWFTracker** en acción; por medio de ella obtenemos muestreos del rendimiento de la película.

## Funcionamiento de Flash Player

Sabemos que cuando creamos una película, esta funciona a modo de *loop* en base a los fotogramas por segundo que hayamos indicado para ella, ya sea por medio de la IDE de Flash o por medio de código. La velocidad a la que se ejecuta este *loop* es definida por los fotogramas por segundo que hayamos indicado para una película: 24 FPS indican una fracción de tiempo en función de un segundo: es decir, "24 veces en un segundo". Por esta razón, sabemos que esta fracción de tiempo representa la velocidad de cuadro (*frame rate*).

Dentro de esta fracción de tiempo, Flash Player realiza dos acciones bien definidas: por un lado, ejecuta código ActionScript, y una vez concluido este proceso, renderiza el contenido visual del escenario. En la medida en que una película en Flash se encuentra corriendo, el motor de Flash Player tratará de mantener la cantidad de FPS que hayamos designado para nuestra película, y **si bien la velocidad no puede ser más rápida de la que hayamos designado, sí puede**

**ser más lenta en caso de que no se llegue a concluir el proceso dentro del tiempo indicado**. Esto puede suceder en aquellos casos en los que utilizamos códigos que exigen un gran proceso por parte del motor o bien cuando empleamos muchos elementos visuales. Independientemente del motivo por el cual se dilate el tiempo de proceso, ya sea por elementos visuales o por código, se reducirá la velocidad de cuadros por segundo con el fin de garantizar que se complete el proceso de manera correcta dentro de cada ciclo. Cuando esto sucede, se reduce la performance de nuestra aplicación: es muy común observar esto en las animaciones. Entonces, uno de nuestros objetivos es lograr mantener el *frame rate* que hayamos designado para nuestra película, y evitar que este se reduzca a fin de ejecutar los procesos de la manera adecuada.

Conseguir que esto no ocurra es nuestro objetivo, y deriva de un mix que se compone de las siguientes variables:

- Utilizar el código adecuado, optimizado para un mejor rendimiento.
- Emplear solamente los elementos visuales que sean necesarios para la aplicación.
- Tomar en consideración el entorno de ejecución de la aplicación.

Teniendo esto en mente, vamos a implementar una solución para nuestro *framework* mediante la cual podremos conocer el rendimiento que está obteniendo Flash Player, considerando tanto los FPS de la película como el uso de la memoria que está llevando.

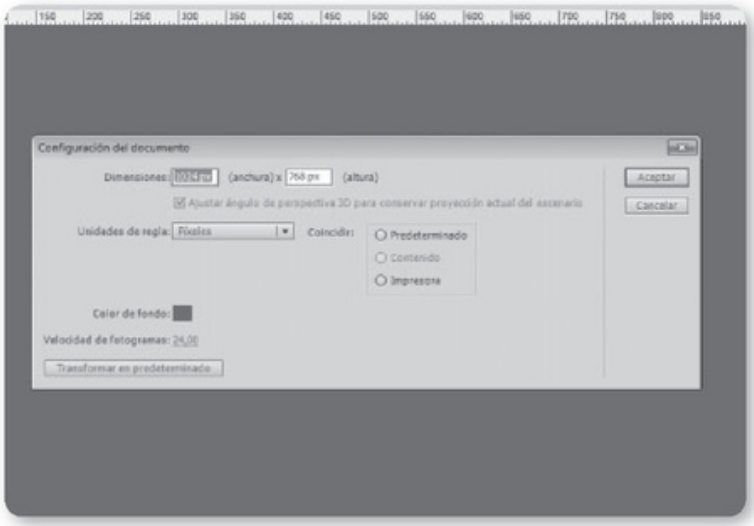



**Figura 2.** Podemos modificar los FPS de la película desde la IDE de Flash, o bien con **`frameRate`** de la clase **`Stage`**.

# Analizar el tracker: SWFTracker.as

Una pequeña aclaración antes de adentrarnos en el código: todos los elementos visuales que componen este pequeño *tracker* de información están generados mediante código. La ventaja de desarrollar este recurso de esta manera es que, al no vincular ningún elemento desde la biblioteca, no tenemos que estar pendientes de ningún elemento visual para garantizar su funcionamiento: los campos de texto son creados de forma dinámica, y los gráficos que se generan son creados por medio de la API de dibujo de Flash, de la cual hablaremos próximamente en este capítulo.

## Constructor de la clase

```
/**
* ...
* SWFTracker
* @param stageReference referencia del escenario para addChild y dibujo desde
    la API
* @param sampling la cantidad de muestras que queremos que tome el tracker
* @param samplingTime el intervalo entre muestra y muestra
* @param indica el tipo de visualizacion: SWFTracker.LINE_VISUALIZATION
    para lineas, SWFTracker.GRAPHIC_VISUALIZATION para recuadros
* @param fpsColor color para muestreo de FPS
* @param fpsAlpha transparencia para muestro de FPS
* @param memoryColor color para muestreo de memoria
* @param memoryAlpha transparencia para muestro de memoria
*/

public function SWFTracker(stageReference:Stage, sampling:uint = 20,
    samplingTime:Number=1, type:String = SWFTracker.LINE_
        VISUALIZATION, fpsColor:uint = 0xff0066, fpsAlpha:Number = 0.4,
            memoryColor:uint = 0xffffff, memoryAlpha:Number = 0.4 ):void {
    this.stageReference = stageReference;
    this.sampling = sampling;
    this.samplingTime = samplingTime;
    this.fpsColor = fpsColor;
```

```
        this.fpsAlpha = fpsAlpha;
        this.memoryColor = memoryColor;
        this.memoryAlpha = memoryAlpha;
        if(type == LINE_VISUALIZATION || type == GRAPHIC_VISUALIZATION) {
            this.type = type;
        }else {
            throw new Error("tipo de visualizacion incorrecto");
        }
        addEventListener(Event.ADDED_TO_STAGE, onAdded, false, 0, true);
    }
```

**Figura 3.** Listado de parámetros que podemos indicarle a la clase. El primero es obligatorio, el resto es opcional.

La clase cuenta con un único parámetro obligatorio, que es el escenario. Necesitamos una instancia de él para poder agregarlo o quitarlo una vez que lo hayamos creado. El resto de los parámetros son de carácter opcional:

## INTERACCIÓN DEL MOUSE

Independientemente de que utilicemos elementos visuales que no cuenten con ningún tipo de interacción (**Sprites** o **MovieClips**), Flash sigue empleando recursos para detectar eventos de mouse sobre estos objetos. La forma de evitarlo es definiendo como *false* la propiedad **mouseChildren** de estos objetos.

### PARÁMETROS DE SWFTRACKER

| PARÁMETRO | DESCRIPCIÓN |
|---|---|
| **stageReference** | Referencia al escenario. |
| **Sampling** | La cantidad de muestras en el tiempo que queremos que evalúe el tracker, tanto para la memoria en uso como para los fotogramas por segundo. |
| **samplingTime** | El tiempo que queremos definir entre muestra y muestra. A menor número, mayor cantidad de muestras. |
| **Type** | Podemos visualizar la información de dos maneras distintas: por medio de una línea o a modo de gráfico. Los parámetros por pasar pueden ser SWFTracker. LINE_VISUALIZATION o SWFTracker .GRAPHIC_VISUALIZATION. |
| **fpsColor** | El color que se le asignará a la representación visual de la información de los fotogramas. |
| **fpsAlpha** | La transparencia que se le asignará a la representación visual de la información de los fotogramas. |
| **memoryColor** | El color que se otorgará a la representación visual de la información de la memoria. |
| **memoryAlpha** | La transparencia que se le otorgará a la representación visual de la información del uso de la memoria. |

**Tabla 1.** Parámetros para la clase **SWFTracker**.

## EVITAR EL USO EXCESIVO DE ANIMACIONES

Si bien la animación genera un aporte significativo en cualquier elemento multimedia, debemos ser medidos con su uso. Existe una importante cantidad de *engines* para animación que, si bien están optimizados para minimizar su peso, incrementan considerablemente el uso de recursos de la memoria si su implementación es abusiva. Debemos ser cuidadosos con este recurso al trabajar para dispositivos móviles, considerando que las prestaciones que nos brindan son limitadas en comparación de un ordenador.

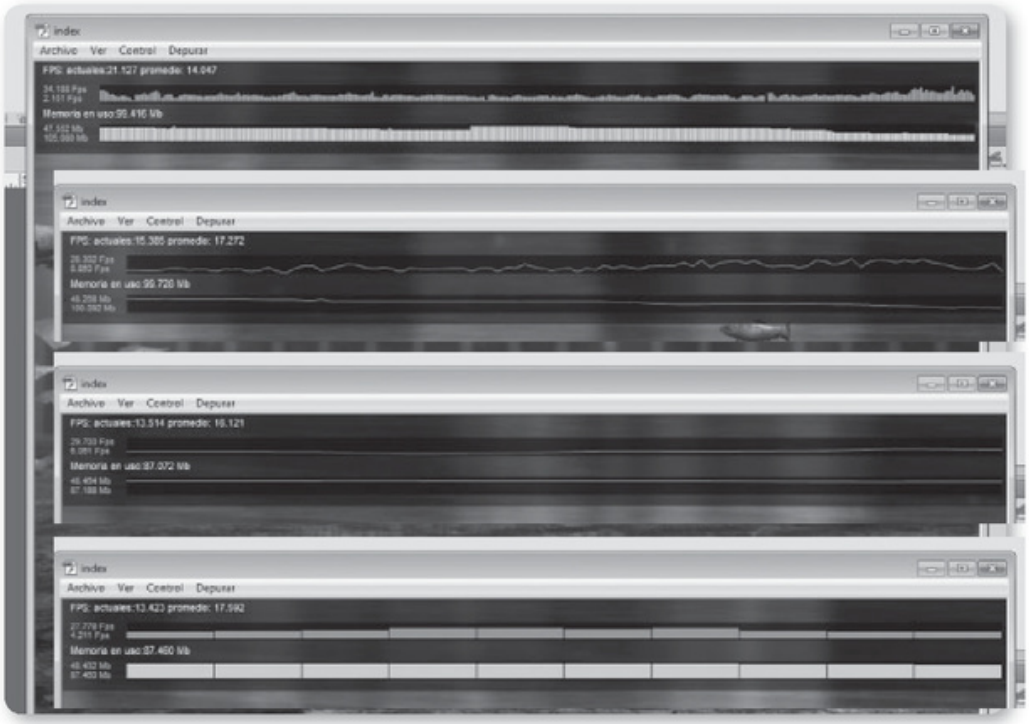



**Figura 4.** Dependiendo de los parámetros que le indiquemos a la clase, podemos obtener distintos tipos de representaciones visuales.

Una vez que igualamos los parámetros que recibe la clase con variables privadas de esta, asignamos un **listener** para saber cuándo se agregó **SWFTracker** al escenario, de la forma siguiente:

```
addEventListener(Event.ADDED_TO_STAGE, onAdded, false, 0, true);

/**
* ...
* onAdded()
* existe referencia al escenario. Inicializar funciones
*/

private function onAdded(e:Event):void {
    removeEventListener(Event.ADDED_TO_STAGE, onAdded);
    stageReference.addEventListener(KeyboardEvent.KEY_DOWN, keyHandler,
        false, 0, true);
```

```
        this.addEventListener(MouseEvent.CLICK, hideTracker, false, 0, true);
        initValues();
        createTextFormat();
        createGUI();
    }
```

El ejecutarse el **onAdded()**, realizamos tres acciones principales:

- Removemos el *listener* anteriormente creado (hablaremos de la importancia de esto en las próximas páginas de este capítulo.
- Asignamos dos *listeners*: uno al escenario para detectar si se están presionando teclas, y otro evento de *mouse* al *tracker* (**this**). (Trataremos estos dos *listeners* en las próximas páginas).
- Llamamos a tres funciones: **initValues()**, **createTextFormat()** y **createGUI()**.

## Inicializar variables: initValues()

La clase denominada **initValues()** les asigna valores a las variables **minFps**, **maxFps**, **minMem**, **maxMem**, y a su vez, define valores para las variables **initTime**, **_intervalTime**, **totalCounter** y **frameCounter**. Haremos uso de estas variables a lo largo del desarrollo:

```
    /**
    * ...
    * initValues()
    * inicializa los valores de FPS y memoria en la app
    */

    private function initValues():void
    {
        minFps = Number.MAX_VALUE;
        maxFps = Number.MIN_VALUE;
        minMem = Number.MAX_VALUE;
        maxMem = Number.MIN_VALUE;
        initTime = _intervalTime = getTimer();
        totalCounter = frameCounter = 0;
    }
```

## Crear formatos de texto para los campos: createTextFormat()

Emplearemos dos formatos de texto (clase **TextFormat**) para asignarles a los campos de texto dinámicos (clase **TextField**). Utilizamos un formato para el título que asignaremos al indicador de fotogramas y de memoria (**titleTextFormat**), y otro para el resto de los campos (**infoTextFormat**):

```
/**
* ...
* createTextFormat()
* crea dos formatos de texto para los campos de texto
*/

private function createTextFormat():void
{
    titleTextFormat = new TextFormat("Arial", 11, 0xffffff);
    infoTextFormat = new TextFormat("Arial", 9, 0xcccccc);
}
```

## Creación de la interfaz gráfica: createGUI()

Dentro de la función **createGUI()** creamos todos los elementos que componen la interfaz visual de nuestra clase. Por un lado, utilizaremos la API de dibujo de Flash tanto para las formas por generar como para los contenedores; por otro, crearemos de manera dinámica los campos de texto sobre los cuales debemos indicar distinto tipo de información.

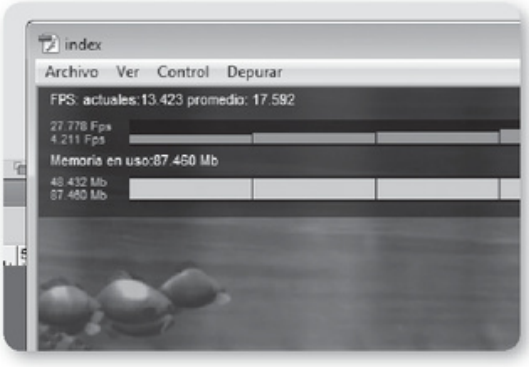



**Figura 5.** Los campos trabajados con `titleTextFormat` presentan un cuerpo tipográfico mayor.

Para la parte visual, utilizaremos la clase **Shape** (veremos la diferencia entre **Shape** y **Sprite** y la funcionalidad de cada uno,

en las próximas páginas). Dentro de la función, crearemos tres formas:

- **container**, para dibujar el *background* de la aplicación.
- **fpsContainer**, para mostrar la información de los fotogramas.
- **memoryContainer**, para la información sobre el consumo de memoria.

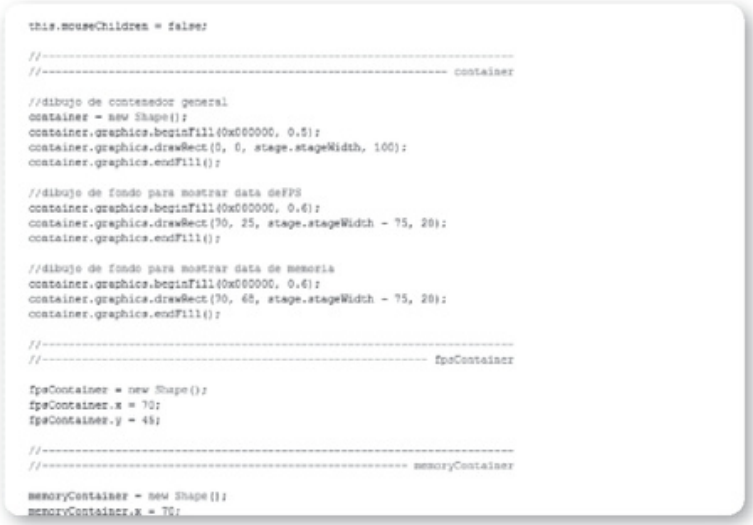


**Figura 6.** Código AS3 mediante el cual generamos los dibujos de las formas.

```
//dibujo de contenedor general
container = new Shape();
container.graphics.beginFill(0x000000, 0.5);
container.graphics.drawRect(0, 0, stage.stageWidth, 100);
container.graphics.endFill();

//dibujo de fondo para mostrar data de FPS
container.graphics.beginFill(0x000000, 0.6);
container.graphics.drawRect(70, 25, stage.stageWidth - 75, 20);
container.graphics.endFill();

//dibujo de fondo para mostrar data de memoria
container.graphics.beginFill(0x000000, 0.6);
container.graphics.drawRect(70, 68, stage.stageWidth - 75, 20);
container.graphics.endFill();

//-----------------------------------------------------
//---------------------------------------- fpsContainer
```

```
fpsContainer = new Shape();
fpsContainer.x = 70;
fpsContainer.y = 45;

//------------------------------------------------
//---------------------------------- memoryContainer
memoryContainer = new Shape();
memoryContainer.x = 70;
memoryContainer.y = 88;
```

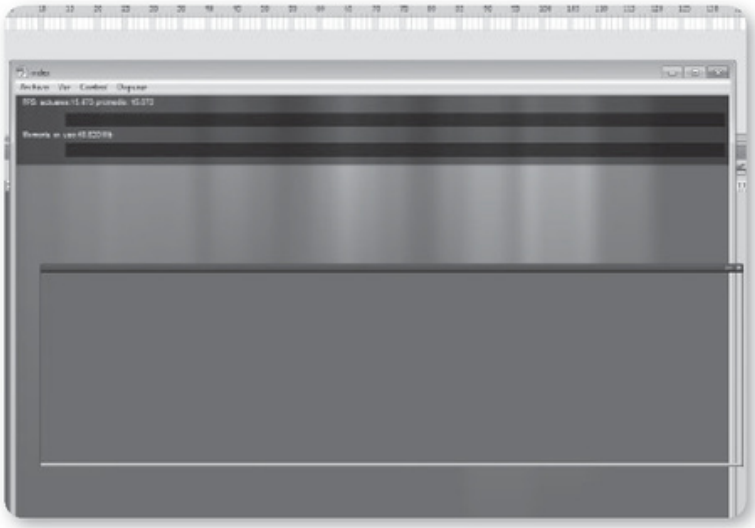


**Figura 7.** En esta imagen podemos ver dibujos realizados dentro del **Shape container**.

Dentro de la misma función creamos todos los campos de texto que vamos a utilizar, tal como vemos a continuación:

```
//------------------------------------------------//-------------------- campos de texto
    para info de FPS

FPSInfoTxt = new TextField();
FPSInfoTxt.autoSize = TextFieldAutoSize.LEFT;
FPSInfoTxt.defaultTextFormat = titleTextFormat;
FPSInfoTxt.x = 7;
FPSInfoTxt.text = "...";
```

```
maxFpsTxt= new TextField();
maxFpsTxt.autoSize = TextFieldAutoSize.LEFT;
maxFpsTxt.defaultTextFormat = infoTextFormat;
maxFpsTxt.x = 7;
maxFpsTxt.y = 22;

minFpsTxt = new TextField();
minFpsTxt.autoSize = TextFieldAutoSize.LEFT;
minFpsTxt.defaultTextFormat =infoTextFormat;
minFpsTxt.x = 7;
minFpsTxt.y = 32;

//-----------------------------------------------------
//------------------------------------ campos de texto para info de memoria

MEMORYInfoTxt = new TextField();
MEMORYInfoTxt.autoSize = TextFieldAutoSize.LEFT;
MEMORYInfoTxt.defaultTextFormat = titleTextFormat;
MEMORYInfoTxt.x = 7;
MEMORYInfoTxt.y = 47;
MEMORYInfoTxt.text = "..."

minMemTxt = new TextField();
minMemTxt.autoSize = TextFieldAutoSize.LEFT;
minMemTxt.defaultTextFormat = infoTextFormat;
minMemTxt.x = 7;
minMemTxt.y = 65;

maxMemTxt = new TextField();
maxMemTxt.autoSize = TextFieldAutoSize.LEFT;
maxMemTxt.defaultTextFormat = infoTextFormat;
maxMemTxt.x = 7;
maxMemTxt.y = 75;
```

Del mismo modo que generamos las formas de manera dinámica haciendo uso de la API de dibujo de Flash, hacemos lo mismo con los campos de texto que empleamos en la interfaz.

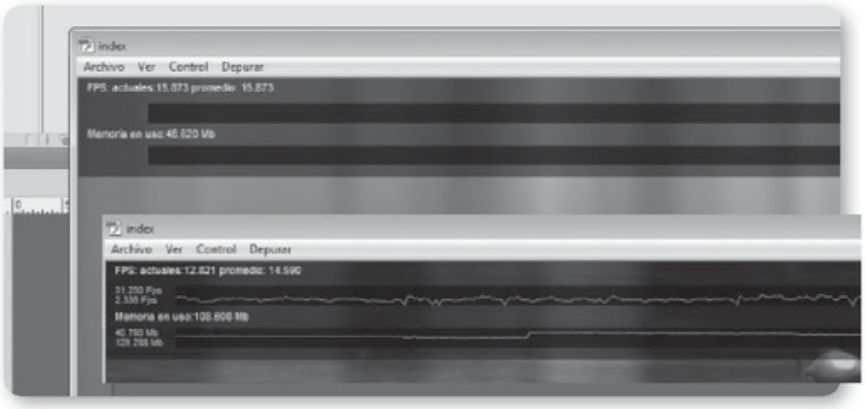



**Figura 8.** En esta imagen podemos ver los campos de texto antes de inicializarse y después de inicializarse.

Y finalmente, los agregamos por medio del método **addChild()** e iniciamos un evento **Event.ENTER_FRAME**:

```
//-----------------------------------------------------
//-------------------- addChild de elementos visuales

addChild(container);
addChild(minFpsTxtBx);
addChild(maxFpsTxtBx);
addChild(minMemTxtBx);
addChild(maxMemTxtBx);
addChild(fpsContainer);
addChild(memoryContainer);
addChild(FPSInfoTxt);
addChild(MEMORYInfoTxt);
stageReference.addEventListener(Event.ENTER_FRAME, checkData, false, 0,
    true);
```

## Comprobar la información: handler checkData()

Además de incrementar en uno los valores de las variables **frameCounter** y **totalCounter**, la función **checkData()** lleva a cabo, por medio de un condicional **if**, la validación, para saber si el intervalo de

tiempo transcurrido es igual o mayor al de muestreo que definimos en el constructor de la clase. En caso de que se dé esta condición, sabemos que tenemos que actualizar la información visual. Entonces, llamamos a la función **updateData()**, que es el método principal de nuestra clase y donde se lleva a cabo la creación de la representación visual de la información, y luego, se llama al método **updateVectors()**. Veremos ambas funciones a continuación.

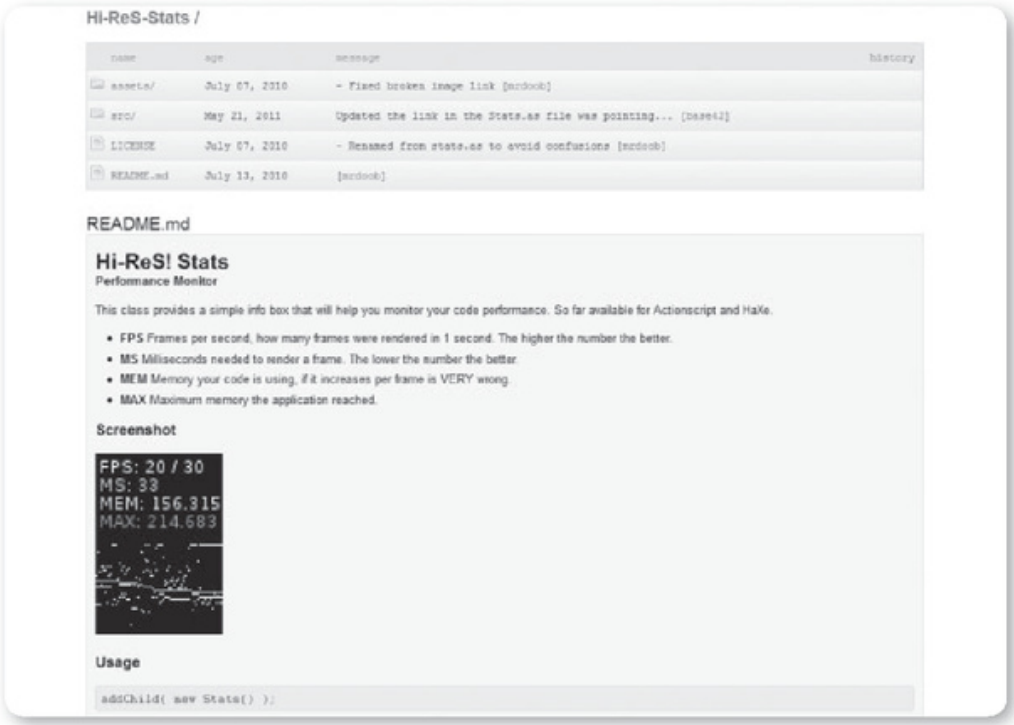



**Figura 9.** Desde el sitio **https://github.com/mrdoob/Hi-ReS-Stats** se puede bajar la clase **Stats de Hi-Rest!**

```
/**
* ...
* checkData()
* se incrementa la cuenta, y si el tiempo de intervalo es mayor
* o igual al tiempo de sampleo, se llama a las funciones que correspondan
* para redibujar los graficos
*/

private function checkData(e:Event) : void {
```

```
        currentTime = getTimer();
        frameCounter++;
        totalCounter++;
        if(intervalTime >= samplingTime) {
            updateData();
            updateVectors();
            _intervalTime = currentTime;
            frameCounter = 0;
        }
    }
```


**Figura 10.** Otra clase muy útil para obtener el rendimiento en FPS y memoria es la empleada en **SOMA**.

## Función updateVectors()

Debemos tener en cuenta que la clase denominada **SWFTracker** contiene dos vectores a los cuales llamamos **fpsHistoryVector** y **memoryHistoryVector**. Dentro del primero, almacenamos el historial de los valores de FPS a los que se estuvo ejecutando la aplicación (**currentFps**), y dentro del segundo, el historial de los valores de memoria empleada en el tiempo transcurrido (**currentMem**).

Necesitamos esta información para crear la representación visual del rendimiento. En vez de almacenar valores dentro del vector por medio del método **push()**, el cual agrega los valores al final del vector, lo hacemos por medio del método **unshift()**, ya que los añade al comienzo:

```
/**
* ...
* updateVectors()
* unshift para reordenar, y pop en caso de que haya mas valores que la cantidad
    de muestras (sampling)
*/

private function updateVectors():void
{
    fpsHistoryVector.unshift(currentFps);
    memoryHistoryVector.unshift(currentMem);
```

**Figura 11.** Al definir en 5 la variable **sampling**, obtenemos solamente cinco muestreos del rendimiento.

Por último, recordemos que, al crear la clase, pasamos como parámetro un valor para la variable **sampling**, que indica la cantidad de muestras que queremos que se lleven a cabo en el tiempo. En caso

## HI-REST! STATS

Desde el sitio web **http://mrdoob.com/blog/post/582** es posible descargar un pequeño visualizador de estadísticas. Al igual que la clase que estamos llevando a cabo en este capítulo, **Stats** evalúa los fotogramas por segundo, su rendimiento y el consumo de memoria que está teniendo la aplicación.

de que los vectores contengan más información que la necesaria para generar las muestras (los vectores son más largos que el valor de la variable **sampling**), quitamos el excedente de información de los vectores por medio del método **pop()**.

```
if(fpsHistoryVector.length > sampling) fpsHistoryVector.pop();
if(memoryHistoryVector.length > sampling) memoryHistoryVector.pop();
```

## Generar la información: updateData()

Dentro del método **updateData()**, realizamos tres acciones principales: por un lado, actualizamos los valores de las variables implicadas en este proceso; luego, actualizamos los campos de texto, y por último, utilizamos la API de dibujo de Flash para generar la representación visual de la información que estamos almacenando. Iremos viendo por partes las fracciones de código más importantes de esta clase.

Por medio del método **min()** de la clase **Math**, evaluamos dos valores y obtenemos el más chico de ambos; y por medio del método **max()**, realizamos la operación contraria. Evaluando los fotogramas actuales y la memoria actual con la cantidad mínima y máxima, tanto de fotogramas como de memoria, obtenemos los valores para las variables **minFps**, **maxFps**, **minMem** y **maxMem**:

```
private function updateData():void {
minFps = Math.min(currentFps, minFps);
maxFps = Math.max(currentFps, maxFps);
minMem = Math.min(currentMem, minMem);
maxMem = Math.max(currentMem, maxMem);
```

### SAMPLING Y SAMPLINGTIME

Si bien la visualización de la información que fue capturando la aplicación se verá de manera más fluida mientras más cantidad de muestras tengamos en menor tiempo, no hay que perder de vista que esto también representa un mayor consumo por parte de nuestra aplicación.

A continuación, evaluamos si el tiempo transcurrido desde que iniciamos el *tracker* es mayor que 1. En ese caso, asignamos los valores de las variables **minFps**, **maxFps**, **minMem** y **maxMem** a los campos de texto **minFpsTxt**, **maxFpsTxt**, **minMemTxt** y **maxMemTxt**. A su vez, asignamos valores para los campos de texto **FPSInfoTxt** y **MEMORYInfoTxt**. Para redondear los decimales en tres, utilizamos el método **toFiexed()**, indicándole 3 como parámetro y a continuación le agregamos el tipo de dato que corresponda: "Fps" para los fotogramas, y "Mb" para el consumo de memoria.

```
if(runningTime >= 1) {
    minFpsTxt.text = minFps.toFixed(3) + " Fps";
    maxFpsTxt.text = maxFps.toFixed(3) + " Fps";
    minMemTxt.text = minMem.toFixed(3) + " Mb";
    maxMemTxt.text = maxMem.toFixed(3) + " Mb";
}
    FPSInfoTxt.text = "FPS: actuales:" + currentFps.toFixed(3) + " promedio: "
        + averageFps.toFixed(3);
    MEMORYInfoTxt.text = "Memoria en uso:" + currentMem.toFixed(3) + "
        Mb";
```

A continuación, viene la parte más interesante del código: por medio de la API de dibujo de Flash, generamos los gráficos con la información que tenemos. Si volvemos unas páginas atrás, veremos que la función **updateData()** se ejecuta cuando el tiempo transcurrido es mayor o igual al tiempo que definimos en la variable **samplingTime**, la cual indica cada cuánto tiempo queremos realizar una nueva muestra. Independientemente de la velocidad, cada vez que entremos en esta función, deberemos eliminar el dibujo que se realizó anteriormente para que no se superpongan. Esto se realiza por medio del método **clear()** de la clase **Graphics**. De esta manera, cada vez que se ejecuten estas dos líneas, se borrará el dibujo anterior:

```
fpsContainer.graphics.clear();
memoryContainer.graphics.clear();
```

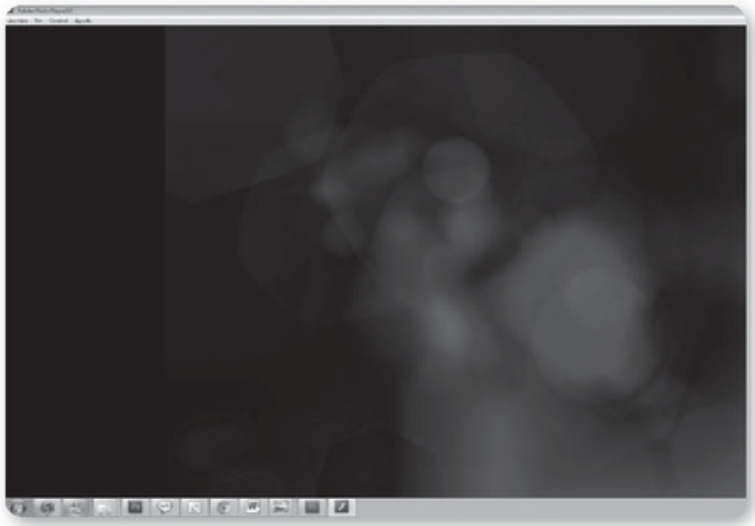


**Figura 12.** Instalación multimedia realizada por **#90ED** utilizando únicamente la API de dibujo proporcionada por Flash.

A continuación, debemos definir las variables para poder hacer el dibujo, el cual se creará dentro de un ciclo **for**. Por un lado, el valor por incrementar se define en la variable **i**, y por otro, necesitamos un valor que indique la cantidad de muestras actuales. Este valor se lo asignamos a la variable **actualSampling**, igualándola al largo del vector **fpsHistoryVector**. Si recordamos, este vector contiene el historial de información que se fue almacenando en el tiempo:

```
var i:int = 0;
var actualSampling:int = fpsHistoryVector.length;
```

Luego creamos tres variables que repercuten directamente en el gráfico que vamos a dibujar: la variable **height** define el alto máximo

## RECURSOS DE LA API DE DIBUJO DE FLASH

Si bien solamente hemos hecho uso de los métodos **drawRect()**, **moveTo()** y **lineTo()** de la API de dibujo de Flash, esta cuenta con una importante cantidad de métodos mediante los cuales podemos generar distintos tipos de formas: **drawElipse()**, **drawCircle()**, **drawRoundRect()**, **drawTriangles()**, entre otras tantas. Las posibilidades que nos brinda la API de dibujo de Flash son inmensas y de gran utilidad para los más diversos desarrollos que podamos encarar.

que tendrá el dibujo, la variable **width** define el ancho (el cual varía dependiendo del tipo de representación), y la variable **inc** indica la diferencia de espacio que hay entre una muestra y otra. Necesitamos esta variable porque el espacio por utilizar y el largo de cada gráfico o línea varía según la cantidad de muestras que queremos visualizar:

```
var height:int = 16;
var width:int = Math.round(stage.stageWidth - 75);
var inc:Number = width / ((type == SWFTracker.GRAPHIC_VISUALIZATION) ?
    sampling : (sampling - 1));
var rectWidth:Number = Math.floor(width / sampling);
```

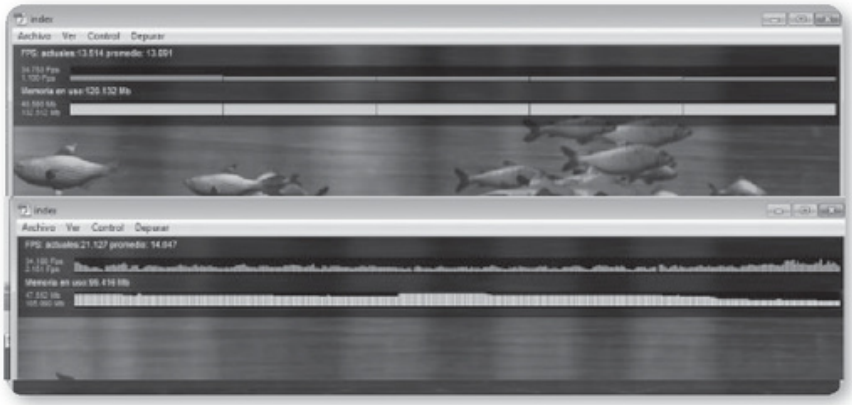


**Figura 13.** La variable denominada **`width`** define el largo, y la variable **`inc`**, la posición en x de cada muestra.

Es necesario tener en cuenta que antes de ingresar al bucle **for**, averiguamos el promedio de FPS y de memoria, y se lo asignamos a las variables **fpsRateTange** y **memoryRateRange**; luego creamos dos variables más (**fpsValue** y **memoryValue**), a las cuales les indicaremos sus valores dentro del ciclo **for**, como vemos a continuación:

```
var fpsRateRange:Number = maxFps - minFps;
var memoryRateRange:Number = maxMem - minMem;
var fpsValue:Number;
var memoryValue:Number;
```

Finalmente, dentro del bucle **for**, realizamos los gráficos. Para enriquecer el ejemplo, antes dijimos que íbamos a emplear dos tipos distintos de visualización. Entonces, dependiendo del tipo de visualización que se definió en el constructor, llevamos a cabo el dibujo de una o de otra manera:

```
for(i = 0;i < actualSampling; i++) {
    fpsValue = (fpsHistoryVector[i] - minFps) / fpsRateRange;
    memoryValue = (memoryHistoryVector[i] - minMem) / memoryRateRange;
    if (type == SWFTracker.GRAPHIC_VISUALIZATION) {
```

**Figura 14.** Representación mediante líneas o recuadros. El tipo de visualización se indica por parámetro en el constructor de la clase.

## Dibujo de rectángulos: drawRect()

En primer lugar, empleamos el método **beginFill()** de la clase **Graphics**, al cual le podemos definir dos valores: el **color** y la **transparencia**. Recordemos que esta información se la pasamos por parámetros al constructor de la clase. En el caso del **Shape fpsContainer**, le asignamos las variables **fpsColor** y **fpsAlpha**; y en el de **Shape memoryContainer**, los valores **memoryColor** y **memoryAlpha**.

A continuación, utilizamos el método **drawRect()** para realizar rectángulos. Podemos asignarle cuatro parámetros: **la posición en x**, **la posición en y**, **el ancho** y **el largo**. Con estas cuatro propiedades definimos el posicionamiento y las dimensiones del rectángulo.

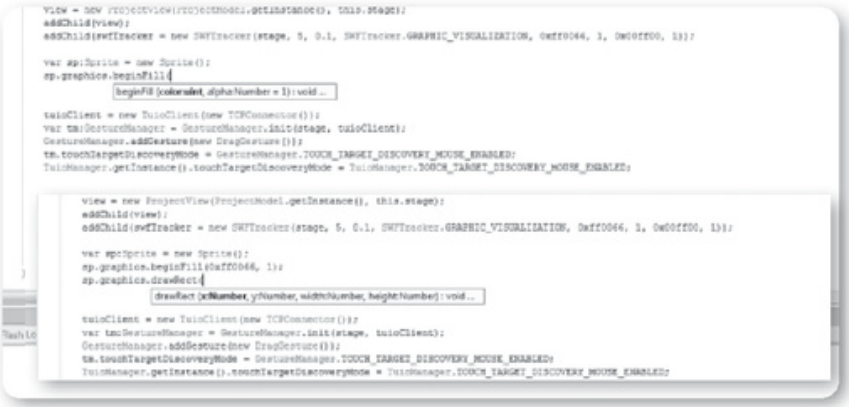


**Figura 15.** Parámetros que se asignarán a los métodos **beginFill()** y **drawRect()** de la clase **Graphics**.

La posición en x se obtiene multiplicando la variable **i** por la variable **inc**: de esta manera, conseguimos que los rectángulos se vayan posicionando uno al lado del otro. La posición **y** la definimos por medio de la variable **MARGIN**, la cual declaramos al comienzo del código y equivale a 2. El largo del recuadro lo obtuvimos previamente dividiendo el largo total del *tracker* por la cantidad de muestras. Recordemos que esto hará que el tamaño de los recuadros varíe según la cantidad de muestras: a mayor número de muestras, menor largo para los rectángulos; y a menor número de muestras, mayor largo para los rectángulos. Por último, el valor más importante es el largo, ya que es el que define el consumo de memoria o el promedio de fotogramas que fuimos obteniendo en el tiempo en cada muestreo: multiplicando el promedio (**fpsValue** en el caso de los fotogramas y **memoryValue** en el caso de la memoria) por el largo que asignamos por defecto (16 píxeles), obtenemos las distintas representaciones que el *tracker* fue logrando, tal como vemos en el siguiente código:

```
fpsContainer.graphics.beginFill(fpsColor, fpsAlpha);
fpsContainer.graphics.drawRect((i * inc), -MARGIN, rectWidth, Math.floor(-
    fpsValue * height));
fpsContainer.graphics.endFill();

memoryContainer.graphics.beginFill(memoryColor, memoryAlpha);
```

```
memoryContainer.graphics.drawRect((i * inc), -MARGIN, rectWidth, Math.floor(-
    memoryValue * height));
memoryContainer.graphics.endFill();
```

**Figura 16.** En esta imagen podemos ver el listado de todos los métodos y propiedades de la API de dibujo de Flash.

## Dibujo de líneas: moveTo() y lineTo()

En caso de que la visualización asignada sea a modo de líneas, ingresamos al **else** del condicional. En las primeras dos líneas de código definimos el estilo de línea que vamos a dibujar por medio del método **lineStyle()** de la clase **Graphics**. El método **lineStyle()** nos permite definir tres valores: el grosor de la línea, su color y la transparencia. Al igual que hicimos en el caso anterior, utilizamos los valores de las variables **fpsColor** y **fpsAlpha** para definir el color y la transparencia de la línea que indica el promedio de FPS, y las variables **memoryColor** y **memoryAlpha** para definir el color y la transparencia en el caso del consumo de memoria:

```
}else {
    fpsContainer.graphics.lineStyle(1, fpsColor, fpsAlpha);
    memoryContainer.graphics.lineStyle(1, memoryColor, memoryAlpha);
```

A diferencia de la modalidad que empleamos anteriormente al utilizar el método **drawRect()**, en el caso de las líneas hay una pequeña

variación: antes de comenzar a dibujar (por medio del método **lineTo()**), debemos movernos a un lugar específico (con el método **moveTo()**). Pensémoslo a modo de analogía con el dibujo real: antes de empezar a dibujar (**lineTo()**), debemos apoyar el lápiz en el papel (**moveTo()**). Para saber qué acción tenemos que realizar, debemos averiguar cuándo se está realizando el primer ciclo, ya que en este deberíamos utilizar el método **moveTo()**. Preguntando en un condicional si la variable **i** es igual a 0, sabemos que estamos en el primer ciclo. En ese caso, utilizamos la función **moveTo()** y le indicamos los dos parámetros que esta acepta: la posición en **x** y la posición en **y** a la cual debe mover el cabezal de dibujo. La posición en **x** es 0, y la posición en **y** se define asignándole los valores de las variables **fpsValue** y **memoryValue**. Si **i** es distinto de 0, empleamos el método **lineTo()** y le indicamos como parámetros para la posición **x** la multiplicación de **i** por el incremento (**inc**), y la multiplicación de la variable **fpsValue** y **memoryValue** por el alto que asignamos por defecto:

```
    (i == 0) ? fpsContainer.graphics.moveTo(0, -fpsValue * height) : fpsContainer.
        graphics.lineTo(i * inc, -fpsValue * height);
    (i == 0) ? memoryContainer.graphics.moveTo(0, -memoryValue * height) :
        memoryContainer.graphics.lineTo(i * inc, -memoryValue* height);
}
```

**Figura 17.** Parámetros de **moveTo()** y **lineTo()**. Podemos recordar su funcionamiento comparándolos con el dibujo real.

Con una sencilla variación obtenemos dos modalidades de visualización. Más importantes que las modalidades son las variables **sampling** y **samplingTime**. Al definir distintos valores para ellas, obtenemos diferentes cantidades de muestras y distintos tiempos entre una y otra.

## Mostrar y ocultar el SWFTracker

Sería normal que, durante el desarrollo, queramos comprobar el funcionamiento de una parte específica, o nos interese tener el **SWFTracker** a mano y tener una forma de fácil acceso, sin salir de nuestro código para hacer un **addChild()** de él. Existen muchas maneras de hacer que su uso sea más sencillo de implementar. Nosotros optamos por ocultarlo mediante un clic, y mostrarlo al escribir "tracker". Recordemos que, al iniciar la clase, asignamos dos *listeners*:

```
stageReference.addEventListener(KeyboardEvent.KEY_DOWN, keyHandler,
    false, 0, true);
this.addEventListener(MouseEvent.CLICK, hideTracker, false, 0, true);
```


**Figura 18.** Implementando unas pocas líneas de código, podemos hacer accesible la clase **SWFTracker** desde el menú contextual.

Cuando hacemos clic en cualquier parte gráfica, llamamos a la función **hideTracker()**. Dentro de ella, la ocultamos y removemos el evento **ENTER_FRAME** para que no se ejecute innecesariamente, ya que no estaríamos haciendo uso de él:

```
/**
* ...
* hideTracker()
* ocultar el tracker
*/
private function hideTracker(e:MouseEvent):void {
    stageReference.removeEventListener(Event.ENTER_FRAME, checkData);
    this.visible = false;
}
```

Por último, para volver a visualizarlo, escribimos **tracker**. Cada vez que se presione una tecla, se ejecutará la función **keyHandler()**. Dentro de ella utilizamos la propiedad **keyCode**, mediante la cual accedemos a un número identificador de cada una de las letras de nuestro teclado. La letra **T** se corresponde con el código 84, la **R** con el 82, la **A** con el 65, la **C** con el 67, la **K** con el 75, y la **E** con el 69. Por medio de un condicional, averiguamos si estas teclas se fueron presionando en orden. Si la primera que se presiona es la **T**, creamos un nuevo vector:

```
private function keyHandler(e:KeyboardEvent):void {

//T R A C K E R
//84 82 65 67 75 69 82

    if (e.keyCode == 84) {
        keyVector = new Vector.<Number>();
        keyVector.push(e.keyCode);
 }
```

## BEGINFILL() Y ENDFILL()

Debemos tener en cuenta que por medio del método **beginFill()**, iniciamos la creación de un dibujo empleando la API. A este método podemos indicarle como parámetros el **color de relleno** y su **transparencia**. Por medio del método **endFill()** finalizamos un dibujo.

Luego realizamos el resto de las validaciones para las teclas:

```
else if (keyVector.length == 1 && e.keyCode == 82) {
    keyVector.push(e.keyCode);
}
else if (keyVector.length == 2 && e.keyCode == 65) {
     keyVector.push(e.keyCode);
}
else if (keyVector.length == 3 && e.keyCode == 67) {
    keyVector.push(e.keyCode);
}
else if (keyVector.length == 4 && e.keyCode == 75) {
    keyVector.push(e.keyCode);
}
else if (keyVector.length == 5 && e.keyCode == 69) {
    keyVector.push(e.keyCode);
}
```

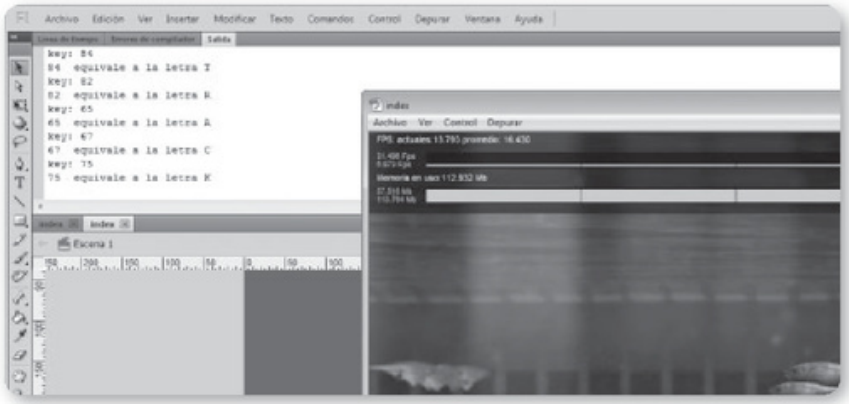


**Figura 19.** En caso de querer averiguar un **keyCode**, el **panel de salida** de Flash puede sernos de ayuda.

De haberse cumplido todas las validaciones, llamamos a **showTracker()**:

```
else if (keyVector.length == 6 && e.keyCode == 82)    {
    showTracker();
```

```
    }
}
```

En **showTracker()** simplemente hacemos visible el objeto y reiniciamos el evento **ENTER_FRAME**:

```
/**
* ...
* showTracker()
* mostrar el tracker
*/
private function showTracker():void {
    stageReference.addEventListener(Event.ENTER_FRAME, checkData, false,
        0, true);
    this.visible = true;
}
```

## Consideraciones finales

Lo verdaderamente importante de esta clase, la razón principal por la cual la hicimos, es porque **nos permite conocer el comportamiento, el rendimiento y la performance que está teniendo nuestra película con el correr del tiempo**. Hacer esto nos sirve para evitar o anticiparnos a los problemas que pueden surgir en la aplicación, para estar más pendientes de los posibles errores que surjan, y para tener una representación real de lo que está ocurriendo con nuestra película, tanto a nivel visual (evaluando los FPS) como a nivel performance, evaluando el consumo de memoria.

### MOSTRAR Y OCULTAR SWFTRACKER

Es necesario tener en cuenta que si bien nosotros estamos mostrando el objeto **SWFTracker** al escribir la palabra "tracker" y ocultándolo al cliquearlo, otra alternativa útil puede ser asignar un botón al menú contextual mediante el cual podamos mostrarlo u ocultarlo.

# Mejorar la performance

A continuación, veremos una serie de *tips* que sirven para mejorar la performance de nuestras aplicaciones. Recordemos: tal vez unos pocos *bytes* no hagan la diferencia si estamos corriendo una aplicación en una computadora, pero seguramente sí la harán en un dispositivo móvil. Es importante no perder de vista el entorno dentro del cual se va a ejecutar la película. Antes de comenzar a ver las distintas mejoras, veamos de qué manera podemos controlar el rendimiento y evaluar los resultados cuando corramos varios procesos distintos.

## Controlar el tiempo: método getTimer()

El método **getTimer()** se emplea para calcular tiempo relativo. Es un método mediante el cual podemos obtener en milisegundos el tiempo transcurrido desde el inicio de la máquina virtual del motor de ejecución de Flash para ActionScript 3.0. Con esta información y almacenándola dentro de una variable, podemos ejecutar un proceso y luego sacar la cuenta de cuánto tiempo le demandó a Flash llevarlo a cabo:

```
var comienzo:Number = getTimer();
//acción mediante AS3
var fin:Number =getTimer() - comienzo;
```

De esta sencillísima forma, podemos acceder a saber cuánto tarda Flash en realizar una acción determinada.

### Grant Skinner: PerformanceTest()

**Grant Skinner** es un reconocido desarrollador que puso a disposición una clase llamada **PerformanceTest()**, una completísima herramienta para evaluar el rendimiento de nuestros códigos. Si bien no emplearemos esta clase en los próximos ejemplos, es de gran utilidad cuando queremos tener información más detallada sobre los procesos que se realizan en una aplicación.

La clase se puede descargar si visitamos la dirección web **http://gskinner.com/blog/archives/2010/02/performancetest.html** y se implementa de la siguiente manera:

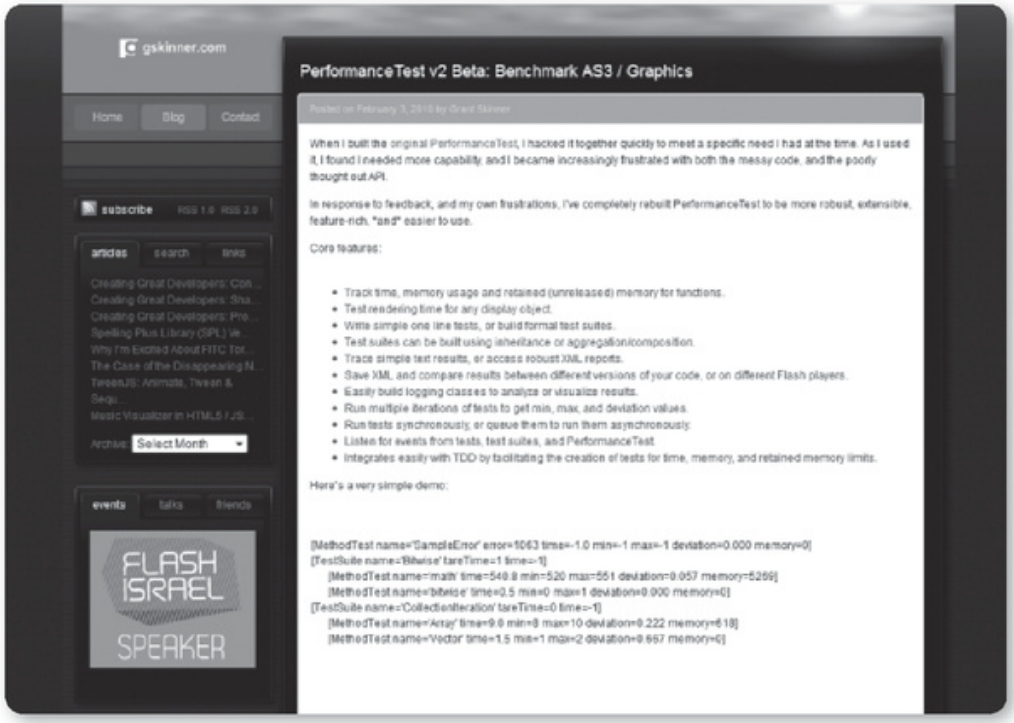



**Figura 20.** En el sitio **`http://gskinner.com`** podemos encontrar información muy valiosa sobre el mundo Flash.

Archivo **performanceTest/index.fla**:

```
//importamos la clase
import com.gskinner.performance.PerformanceTest;

//creamos una instancia
var pt:PerformanceTest;

//definimos dos funciones a testear:
//testArray();

function testArray():void{
```

```
    var testArray:Array = new Array();
    for (var i:int = 0; i< 1000000; i++)
    {
        testArray[i] = i;
    }
}

//testVector();

function testVector():void{
    var testVector:Vector.<Number> = new Vector.<Number>();
    for (var j:int = 0; j< 1000000; j++)
    {
        testVector[j] = j;
    }
}
//obtenemos instancia
pt = PerformanceTest.getInstance();
//ejecutamos las dos funciones:
trace(pt.runSimpleTest(testArray, null, "testArray", 5, 1));
trace(pt.runSimpleTest(testVector, null, "testVector", 5, 1));
```

Nos interesan las últimas dos líneas del código: por medio del método **runSimpleTest()** de la clase, podemos evaluar de forma mucho más detallada el rendimiento de una función, el tiempo que tarda en ejecutarse y el consumo de memoria.

A nosotros nos interesa demostrar solamente la diferencia de rendimiento y de performance, por lo que en los próximos ejemplos

## VALOR DE GETTIMER()

Flash emplea dos máquinas virtuales para procesar ActionScript: **AVM1** para procesar AS1 y AS2, y **AVM2** para procesar AS3. El valor devuelto por el método **getTimer()** es distinto al obtenerlo de diferentes máquinas virtuales. Esto se debe a que ambas tienen distinta fecha de inicio.

haremos unas sencillas cuentas manualmente, pero esta clase es de inmensa ayuda cuando queremos procesar información más compleja y detallada, y por esta razón debemos tenerla en cuenta.

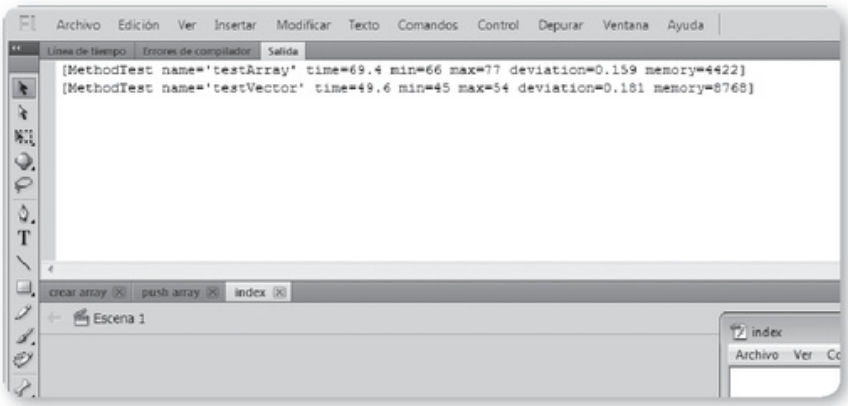



**Figura 21.** Al hacer un **`trace()`** desde Flash, podemos visualizar la completa información que nos devuelve la clase.

# Vectores vs. Arrays

La clase **Vector** se incluyó en la versión 10 de Flash Player, y es considerablemente más rápida que la clase **Array** al realizar operaciones. De todos modos, nos sirve solo en aquellos casos en los que los elementos por agregar dentro son del mismo tipo (por ejemplo, todos objetos, o todas cadenas de texto, o todos números). La mejor forma de corroborarlo es por medio de un sencillo ejemplo, pero antes veamos de qué modo evaluar el tiempo que le lleva a Flash realizar un proceso.

## Testear rendimento de vectores y testear arrays

Por medio del siguiente código, realizaremos la misma operación de almacenamiento en un **Vector** y en un **Array**, y calcularemos cuánto tiempo demora cada proceso:

```
var testArray:Array = new Array();
var comienzoArray:Number = getTimer();
for (var i:int = 0; i< 1000000; i++)
```

```
{
testArray[i] = Math.random();
}
trace("tiempo para almacenar 1.000.000 valores en array:", getTimer() -
    comienzoArray);

var testVector:Vector.<Number> = new Vector.<Number>();
var comienzoVector:Number = getTimer();
for (var j:int = 0; j< 1000000; j++)
{
testVector[j] = Math.random();
}
trace("tiempo para almacenar 1.000.000 valores en array:", getTimer() -
    comienzoVector);
```

En el panel de salida de Flash, veremos los siguientes mensajes:

```
tiempo para almacenar 1.000.000 valores en array: 323
tiempo para almacenar 1.000.000 valores en vector: 142
```

Ante la misma cantidad de valores para almacenar, hacerlo dentro de un array demoró 323 milisegundos, y dentro de un vector, 142.

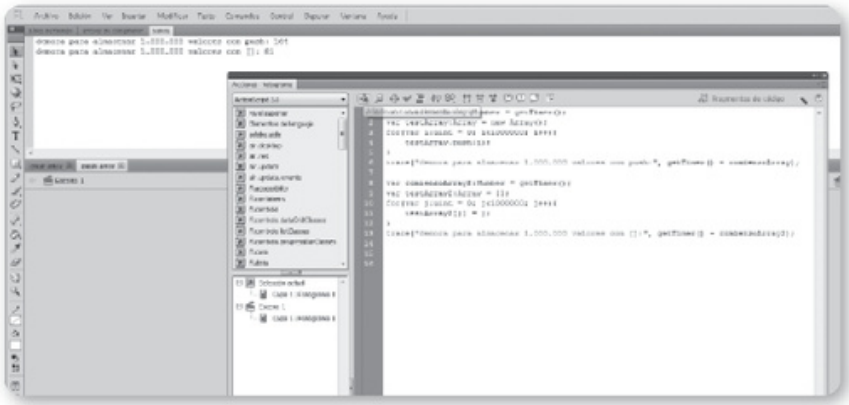


**Figura 22.** La diferencia en milisegundos entre procesos muestra el superior rendimiento de la implementación de vectores.

## Optimizar el almacenamiento de valores en vectores y arrays

Aún podemos seguir optimizando el uso de vectores y arrays. Flash cuenta con **push()**, mediante el cual agregamos valores dentro de un array. Existe una forma aún más rápida de hacerlo: indicando la posición en la cual queremos agregar el valor: **array[largoDelArray] = objetoAColocar**.

```
var comienzoArray:Number = getTimer();
var testArray:Array = new Array();
for(var i:uint = 0; i<50000; i++){
    testArray.push(i);
}
trace("demora para almacenar valor con push:", getTimer() - comienzoArray);

var comienzoArray2:Number = getTimer();
var testArray2:Array = [];
for(var j:uint = 0; j<50000; j++){
    testArray2[j] = j;
}
trace(testArray2.length);
trace("demora para almacenar valor con []:", getTimer() - comienzoArray2);
```

Al ejecutar el código y realizar las cuentas del tiempo transcurrido utlizando la propiedad **getTimer()**, obtenedremos los siguientes mensajes en el panel de salida de Flash:

```
demora para almacenar valor con push: 7
demora para almacenar valor con []: 3
```

### VIDEOTUTORIAL SOBRE PERFORMANCE

Desde la URL **www.gotoandlearn.com/play.php?id=115** es posible acceder a un videotutorial de **Lee Brimelow** en el cual se explica el uso de la clase **PerformanceTest** de **Grant Skinner**. El tutorial está en inglés, y los archivos pueden descargarse desde la misma página.

A la segunda modalidad que utilizamos, le toma menos de la mitad del tiempo almacenar 50.000 valores.

## Optimizar la creación de elementos (arrays, objetos, etcétera)

ActionScript tarda considerablemente menos en crear elementos empleando corchetes **[]** en lugar de **new**. Veamos dos ejemplos creando objetos y creando arrays:

```
var comienzoArray:Number = getTimer();
for(var i:uint = 0; i<100000; i++){
    var testArray:Array = new Array();
    var testObject:Object = new Object();
}
trace("demora para crear elementos con New:", getTimer() - comienzoArray);

var comienzoArray2:Number = getTimer();
for(var j:uint = 0; j<100000; j++){
    var testArray2:Array = [];
    var testObject2:Object = [];
}
trace("demora para crear elementos con []:", getTimer() - comienzoArray2);
```

Si vamos al panel de salida, encontraremos el siguiente mensaje:

```
demora para crear elementos con New: 162
demora para crear elementos con []: 95
```

# Concatenar texto: método appendText()

La clase **TextField** contiene el método **appendText()**, el cual se emplea para concatenar cadenas de texto. Muchos desarrolladores suelen concatenar strings utilizando el operador **+**. Evaluemos el tiempo de una operación y de otra y veamos la diferencia en rendimiento de ambos métodos utilizando el código que mostramos a continuación:

```
import flash.text.TextField;
import flash.ui.Keyboard;
import flash.utils.Timer;

var comienzoField1:Number = getTimer();
var field1:TextField = new TextField();

for(var i:uint = 0; i<5000; i++){
    field1.text+= " texto "
}
trace("demora para suma+:", getTimer() - comienzoField1);
var comienzoField2:Number = getTimer();
var field2:TextField = new TextField();
for(var j:uint = 0; j<5000; j++){
    field2.appendText(" texto ");
}
trace("demora para appendText:", getTimer() - comienzoField2);
```

Resultado en el panel de salida:

```
demora para suma+: 4424
demora para appendText: 3954
```

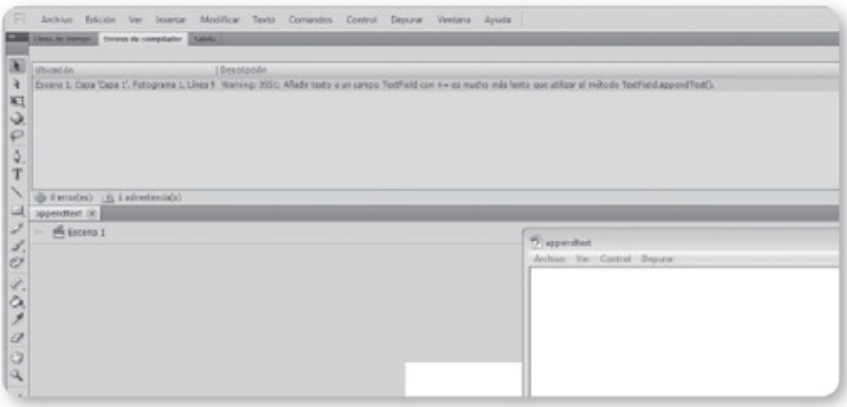


**Figura 23.** Flash nos indica en el panel **`Errores de compilador`** que el método **`appendText()`** es más eficiente.

# Dibujo por código vs. dibujo vectorial

En aquellos casos en los que debemos crear formas sencillas (rectángulos, triángulos, círculos), el uso de la API reduce considerablemente el peso final de un archivo. Veamos un ejemplo.

## Dibujo desde la IDE

Una vez creado un rectángulo de 200 x 200 píxeles, lo convertimos en MovieClip, y lo vinculamos en la librería bajo el nombre **Sq**:

```
var sq:Sq = new Sq();
sq.x = sq.y = 20;
addChild(sq);
```

Si ejecutamos la película, obtendremos el siguiente resultado:

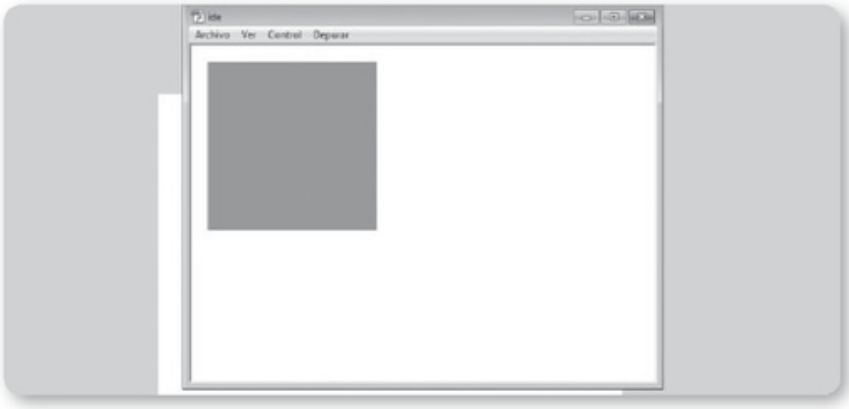


**Figura 24.** Una vez ejecutado el código, se coloca el rectángulo en las coordenadas **x=20** e **y=20** de la película.

## Dibujo por código utilizando la API de dibujo de Flash

Al usar la API de dibujo de Flash, debemos crear la forma por medio de código. Si empleamos las siguientes líneas, le damos forma a un rectángulo de las mismas dimensiones y del mismo color que el creado desde la IDE. También lo posicionamos en las mismas coordenadas **x** e **y**:

```
import flash.display.Sprite;

var sq:Sprite = new Sprite();
sq.x = sq.y = 20;
sq.graphics.beginFill(0xff0066, 1);
sq.graphics.drawRect(0, 0, 200, 200);
sq.graphics.endFill();

addChild(sq);
```

Al exportarlo, veremos que el SWF resultante es visualmente igual al que hemos creado desde la IDE de Flash.

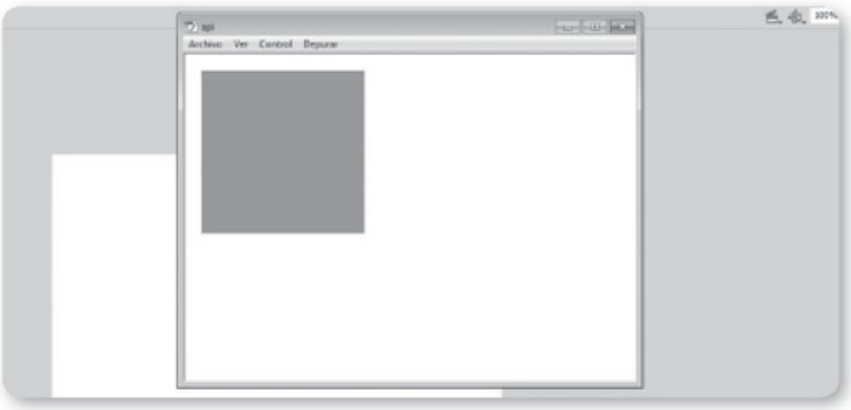


**Figura 25.** Rectángulo creado por medio de la API de dibujo de Flash. En apariencia, es idéntico al generado desde la IDE.

Ahora bien, si comparamos los pesos de ambos archivos, veremos que el **.SWF** que contiene el rectángulo dibujado por medio de la API de dibujo de Flash es considerablemente más liviano que el **.SWF** que contiene el cuadrado creado desde la IDE y vinculado desde la librería.

Una técnica no es un reemplazo de la otra: en muchas ocasiones, aun siendo conscientes de la diferencia en tamaños, sigue siendo más útil y práctico dibujar desde la IDE de Flash. Lejos de tomar partido por una modalidad de dibujo, lo ideal es aplicar la solución que corresponda al desarrollo que corresponda. Este *tip* es importante para aquellos desarrolladores que tengan particular interés en la creación de

aplicaciones para dispositivos móviles, donde los pesos de los archivos realmente deben tomarse en consideración.

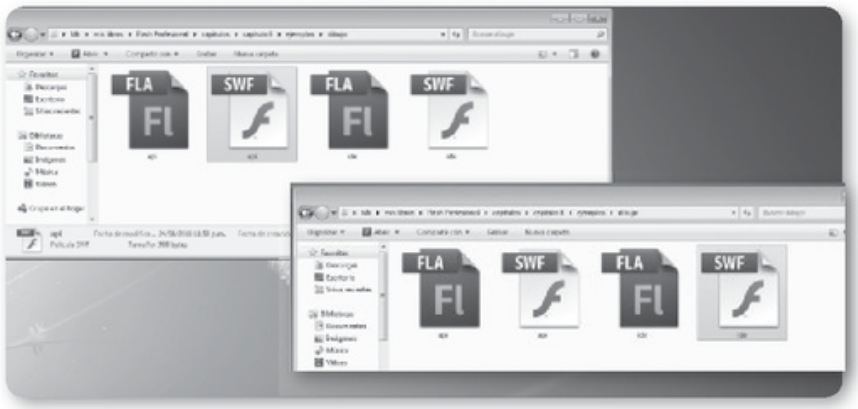



**Figura 26.** Visualmente ambas películas son iguales, pero podemos ver la diferencia en tamaños: **957 bytes** contra **988**.

# Reducir el consumo de memoria

A continuación, veremos de qué manera podemos reducir el consumo de memoria en nuestras aplicaciones.

## Elección del display object adecuado

Flash cuenta con una importante cantidad de *Display Objects*. Solemos hacer uso de dos de ellos con mucha frecuencia: la clase **Sprite** y la clase **MovieClip**. Veamos en qué caso resulta conveniente usar cada uno.

### Emplear Sprites en lugar de MovieClips

Un **Sprite**, a diferencia de un **MovieClip**, no contiene en su interior una línea de tiempo. Esto ahorra unos pocos *bytes* a nuestra aplicación. Flash cuenta con un método **getSize()**, que indica cuántos bytes consume un objeto en nuestra memoria. Ejecutemos el siguiente código:

```
trace("espacio en memoria para Shape:", getSize(new Shape()));
trace("espacio en memoria para Sprite:", getSize(new Sprite()));
trace("espacio en memoria para MovieClip:", getSize(new MovieClip()));
```

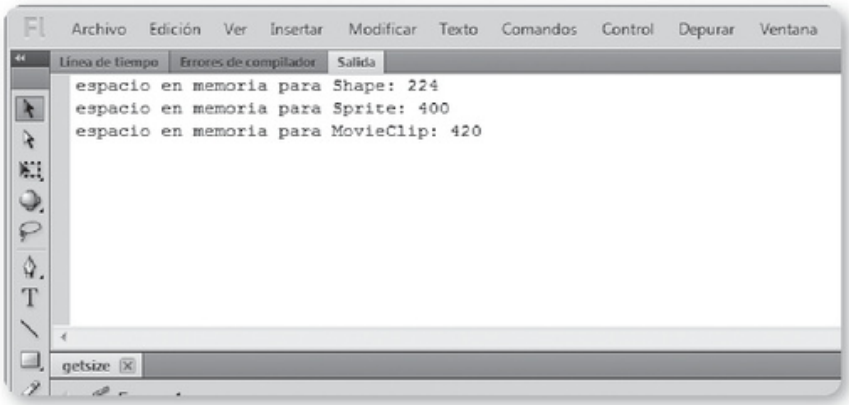



**Figura 27.** Si ejecutamos el código en la película, obtendremos los valores del espacio que ocupa cada uno en memoria.

Veamos otro sencillo ejemplo. Si creamos una clase y la extendemos a **Sprite** en lugar de hacerlo a **MovieClip**, veremos que el archivo **.SWF** resultante pesa menos. Son pocos *bytes* de diferencia, pero no perdamos de vista que se trata de una única clase. Pensemos en la cantidad que hemos utilizando a lo largo de todo este libro.

**Figura 28.** Diferencia de pesos entre dos archivos idénticos, uno extendido a **`Sprite`** y otro a **`MovieClip`**.

## Empleo de Shapes en vez de Sprites

Una forma (**Shape**) ocupa menos espacio aún que un **Sprite**, pero no la podemos utilizar en casos que requieran interacción. De todos modos, sigue siendo una buena alternativa en muchas ocasiones: lo importante es saber cuándo emplear un objeto en lugar de otro. Al crear las formas del visualizador de información, lo hicimos sobre **Shapes** y no, sobre **Sprites**. Esto se debe, justamente, al hecho de que los elementos visuales generados no contienen ningún tipo de interactividad.


**Figura 29.** Entre un **Shape** y un **Sprite**, la diferencia es de 176 bytes. Si utilizamos 500 **Shapes** nos ahorramos unos 176 x 500.

# Declaración y tipo de variables

Es normal que, al crear un bucle **for**, declaremos dentro de la misma sentencia la variable que vamos a utilizar:

```
for(var i:uint = 0; i<5000000; i++){
```

Si ejecutamos el siguiente código, veremos que la performance de Flash mejora al declarar la variable fuera:

```
var comienzo1:Number = getTimer();
var testArray:Array = new Array();
for(var i:int = 0; i<5000000; i++){
```

```
    testArray[i] = i;
}
trace("declaracion dentro del for:", getTimer() - comienzo1);


var comienzo2:Number = getTimer();
var testArray2:Array = [];
var j:int;
for(j = 0; j<5000000; j++){
    testArray2[j] = j;
}
trace("declaracion fuera del for:", getTimer() - comienzo2);
```

Panel de salida:

```
declaracion dentro del for: 423
declaracion fuera del for: 369
```

## Tipo de dato para variables numéricas

Tanto la variable **i** como la variable **j** que empleamos en el ejemplo anterior son datos del tipo **int**. Tanto **int** como **uint** nos permiten crear valores de hasta 32 bits, y en el caso del tipo de dato **Number**, el valor asciende a 64 bits. Si en el ejemplo anterior modificamos el tipo de dato de ambos valores por **Number** y volvemos a ejecutar la película, el panel de salida nos mostrará la siguiente información:

```
declaracion dentro del for: 538
declaracion fuera del for: 461
```

En ambos casos, el tiempo que le llevó a Flash concluir la operación se incrementó en más de 100 milisegundos. Debemos emplear variables del tipo **Number** cuando realmente necesitemos hacer uso de ellas: en general, esto sucede cuando debemos emplear valores que ocupen más de 32 bits, o bien al trabajar con valores de coma flotante.

# Reducir el uso de la CPU

También contamos con una serie de recursos que nos permiten reducir considerablemente el uso de la CPU. Estas buenas prácticas resultan útiles ante cualquier tipo de desarrollo.

## Foco en el escenario: Event.ACTIVATE y Event.DEACTIVATE

Por medio de los eventos **Event.ACTIVATE** y **Event.DEACTIVATE**, podemos saber cuándo se está haciendo foco sobre la película y cuándo no. Contando con esta información, la podemos utilizar para modificar en tiempo de ejecución la cantidad de FPS de la aplicación y, así, optimizar el uso de los recursos de memoria. Su implementación es muy sencilla:

```
import flash.events.Event;

stage.addEventListener(Event.ACTIVATE, setAppFrameRate, false, 0, true);
stage.addEventListener(Event.DEACTIVATE, resetappFrameRate, false, 0, true);

function setAppFrameRate (e:Event):void
{
  stage.frameRate = 24;
    frameTateTxt.text = "FPS: " + stage.frameRate;

    }
}
```

### IGUALAR ELEMENTOS VISUALES A NULL

Es necesario tener en cuenta que independientemente de que eliminemos los elementos visuales que ya no necesitamos por medio del método denominado **removeChild()** y les quitemos sus respectivos *listeners*, estos objetos siguen permaneciendo en memoria. Si queremos que el *garbage collector* realmente los elimine, debemos igualarlos a **null**. Ejemplo: **mySprite = null**

```
function resetappFrameRate(e:Event):void
{
    stage.frameRate = 0;
    frameTateTxt.text = "FPS: " + stage.frameRate;
}
```


**Figura 30.** Si se está haciendo foco o no en la aplicación, modificamos la propiedad **`frameRate`** del escenario (**`stage`**).

# Quitar elementos visuales y remover listeners

No nos queda otra opción: es necesario quitar tanto los elementos visuales, como los *listeners*, cuando estos dejan de utilizarse. Todos aquellos elementos visuales que no eliminamos siguen formando parte del *display list* (listado de los elementos visuales de nuestra aplicación), y a su vez, aquellos *listeners* que no quitamos, siguen utilizando recursos de la memoria. Es importante ir formando una costumbre respecto a este tema y eliminar todo lo que no utilizamos: es sorprendente la mejora en el rendimiento y la performance cuando somos cuidadosos respecto a este punto. En la carpeta **remove** de los ejemplos se encuentran los archivos por utilizar.

Clase **Main.as**: al inicializarse, creamos un bucle **for** que realiza 1000 ciclos y, por lo tanto, crea 1000 instancias de la clase **Ball**. A continuación, las añade al escenario, tal como vemos a continuación:

```
package
{
    import flash.display.Sprite;

    import Ball;
    /**
 * ...
 * @author #90ED
 */

    public class Main extends Sprite
    {
        private var ball:Ball;
        public function Main():void {

            for (var m:uint = 0; m < 500; m++) {
                ball = new Ball();
                stage.addChild(ball);
            }
        }
    }
}
```

La clase **Ball** es nada más y nada menos que un círculo creado mediante la API de dibujo de Flash:

```
public function Ball():void {
    this.graphics.beginFill(0xff0066, Math.random()/2);
    this.graphics.drawCircle(0, 0, 10 * Math.random());
    this.graphics.endFill();
    addEventListener(Event.ADDED_TO_STAGE, onAdded, false, 0, true);
}
```

La línea que en verdad nos interesa es la 5: lo primero que debemos averiguar es cuándo se agregó el elemento al escenario. Para esto, asignamos el siguiente *listener*:

```
addEventListener(Event.ADDED_TO_STAGE, onAdded, false, 0, true);
```

Cuando se ejecute la función denominada **onAdded()** en respuesta al evento, inmediatamente lo borramos: no tiene nada más que hacer en nuestro código, tal como sigue:

```
private function onAdded(e:Event):void {
    trace("se agrego el elemento al escenario: eliminar listener: iniciar
        animacion");
    removeEventListener(Event.ADDED_TO_STAGE, onAdded);
    addEventListener(Event.ENTER_FRAME, setDirection, false, 0, true);
}
```

Dentro de la misma función, asignamos un evento **ENTER_FRAME**, mediante el cual vamos a animar cada uno de los círculos creados por la clase, como lo muestra el siguiente código:

```
private function setDirection(e:Event):void {
    this.x += Math.random() * 10;
    this.y += Math.random() * 10;
```

Al incrementar gradualmente su posición tanto en **x** como en **y**, llegado el momento, estos elementos van a salir de escena. En ese caso, no tienen por qué seguir ocupando espacio y memoria dentro de nuestra aplicación. Una posible solución es preguntar por su posición: si esta superó el largo del escenario o el ancho, eliminamos el *listener*, y quitamos el elemento del escenario:

## LISTENERS Y GARBAGE COLLECTOR

Al crear *listeners*, definimos como *true* el último parámetro, que por defecto es *false*. Este último parámetro (**useWeakReference**) determina si la referencia del *listener* es fuerte o débil. Al setearlo en *false* nos garantizamos que, al pasar el *Garbage Collector*, el *listener* sea quitado completamente de memoria.

```
if (this.x + this.width > stage.stageWidth + 100|| this.y + this.height > stage.
    stageHeight + 100) {
    removeEventListener(Event.ENTER_FRAME, setDirection);
    stage.removeChild(this);
    trace("se removio el listener y el elemento");
    }
}
```

De no hacer esto, nuestra aplicación seguirá siendo funcional, pero tendremos 1000 eventos **ENTER_FRAME** ejecutándose simultáneamente sin fin alguno. El evento **ENTER_FRAME** se ejecuta tantas veces como fotogramas por segundo tenga nuestra aplicación.

A un promedio de 24 FPS y 1000 eventos innecesarios, son 24.000 por segundo que no cumplen función alguna.

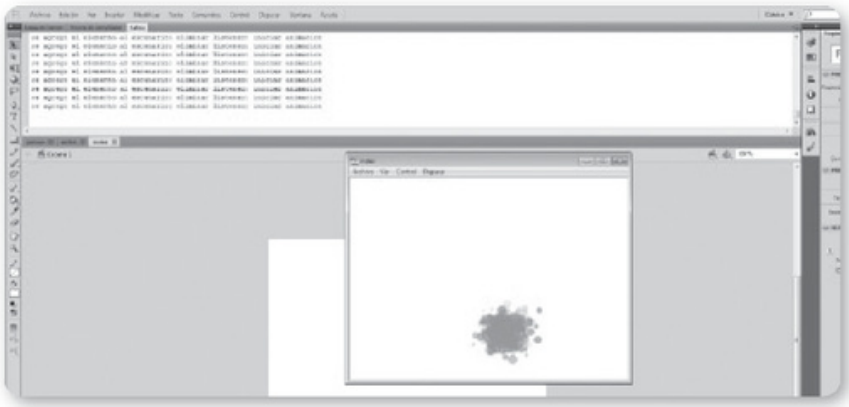


**Figura 31.** A medida que cada instancia de **Ball** sale de pantalla, se la remueve del escenario y se quita su listener.

## CALIDAD DE LA PELÍCULA

Por medio de la clase **StageQuality**, podemos modificar la calidad de nuestra película (las opciones son **LOW**, **MEDIUM**, **HIGH**, **BEST**). Bajar la calidad puede sernos de utilidad al estar ante un desarrollo pensado para un dispositivo móvil, ya que así ocupará menos recursos y procesos del dispositivo, y sin que se modifique drásticamente la calidad visual.

# Intervalos: clase Timer y evento ENTER_FRAME

Tanto la clase **Timer** como el evento **ENTER_FRAME** se utilizan para ejecutar funciones cada determinado intervalo de tiempo. El evento **ENTER_FRAME**, como su nombre lo indica, opera cada vez que la película accede a un nuevo fotograma. Si los FPS de nuestra película son 24, de ejecutarse un evento **ENTER_FRAME**, este lo hará 24 veces en un segundo. En cuanto a los *timers*, la frecuencia con la que se ejecutan es definida por el usuario.

Hasta aquí hemos analizado los diferentes recursos que tenemos a nuestra disposición para lograr que nuestros desarrollos se encuentren optimizados. Es muy importante tener en cuenta cada uno de los consejos mencionados en este capítulo ya que serán la diferencia entre una aplicación web exitosa y otra que pase inadvertida dado su bajo nivel de rendimiento.

## RESUMEN

La optimización de los contenidos no es un factor que los desarrolladores suelan tener en cuenta. Años atrás, no hubiésemos imaginado que necesitaríamos programar y diseñar para dispositivos con menor capacidad de proceso, menor resolución y menores prestaciones, pero hoy esto es una realidad: los dispositivos móviles nos exponen a este tipo de limitaciones, y para aprovechar esta oportunidad de mercado, debemos ser conscientes del uso que hacemos de nuestros recursos. Las buenas prácticas repercuten siempre de manera positiva, independientemente del tipo de desarrollo.

# Actividades

## TEST DE AUTOEVALUACIÓN

1. ¿Cuál es la diferencia entre un **Vector** y un **Array**?
2. ¿Cuál es más rápido? ¿De qué manera se puede optimizar aún más el rendimiento de un array o de un vector?
3. ¿Cuál es la diferencia entre el dibujo vectorial y el dibujo implementando la API de dibujo de Flash? ¿En qué caso se debe emplear una modalidad de dibujo, y en qué caso la otra? ¿Qué modalidad brinda una mejor performance?
4. ¿Para qué se utilizan las clases **Shape**, **Sprite** y **MovieClip**? ¿Cuál es la más pesada y cuál la más liviana?
5. ¿Para qué sirven los eventos **Event.ACTIVATE** y **Event.DEACTIVATE**? ¿Con qué propiedad del **Stage** modificamos la velocidad de los fotogramas?
6. ¿De qué manera funciona **Flash Player**?
7. ¿Cuál es la diferencia entre el evento **Event.ENTER_FRAME** y la clase **Timer()**? ¿En qué casos conviene implementar uno y en qué casos el otro?
8. ¿Por qué es necesario eliminar los objetos que ya no se utilizan?
9. ¿Por qué es necesario eliminar los listeners que ya no estamos empleando en nuestra película? ¿Por qué debemos eliminar los eventos **Event.ENTER_FRAME** que no utilizamos?
10. ¿Para qué sirve el método **getTimer()** de Flash?

## ACTIVIDADES PRÁCTICAS

1. Agregue las funcionalidades que crea necesarias para la clase **SWFtracker**.
2. Haga uso de **PerformanceTest** de **Grant Skinner** y testee con ella los ejemplos que hemos utilizado en este capítulo.
3. Haga que **SWFTracker** sea accesible desde un botón del menú contextual.
4. Averigüe respecto de otras formas de obtener la información necesaria sobre fotogramas y memoria, e impleméntelas en algún desarrollo.
5. Averigüe si existe alguna posibilidad de optimizar los procesos de carga de estructuras XML, utilizando alguna metodología alternativa.

# *09 –

# Comunicar desarrollo con el framework

A lo largo de este libro, hemos ido creando nuestro propio espacio de trabajo, y ahora llegó el momento de ponerlo en funcionamiento a través de un caso práctico. En las próximas páginas veremos cómo articular un proyecto real a fin de apreciar los beneficios de haber pensado previamente en cada detalle que conforma nuestro framework.

# Estructura del proyecto

En capítulos previos pudimos revisar los fundamentos de nuestra modalidad de trabajo: hemos visto cómo crear un nuevo proyecto de FlashDevelop, cómo organizar la estructura del archivo XML (**site.xml**) y cómo hacer funcional nuestro propio *framework*. Por cuestiones metodológicas, no vamos a repetir aquí los pasos que componen este proceso, pero sí es imprescindible tener a mano la carpeta que contiene el proyecto que realizaremos a continuación.

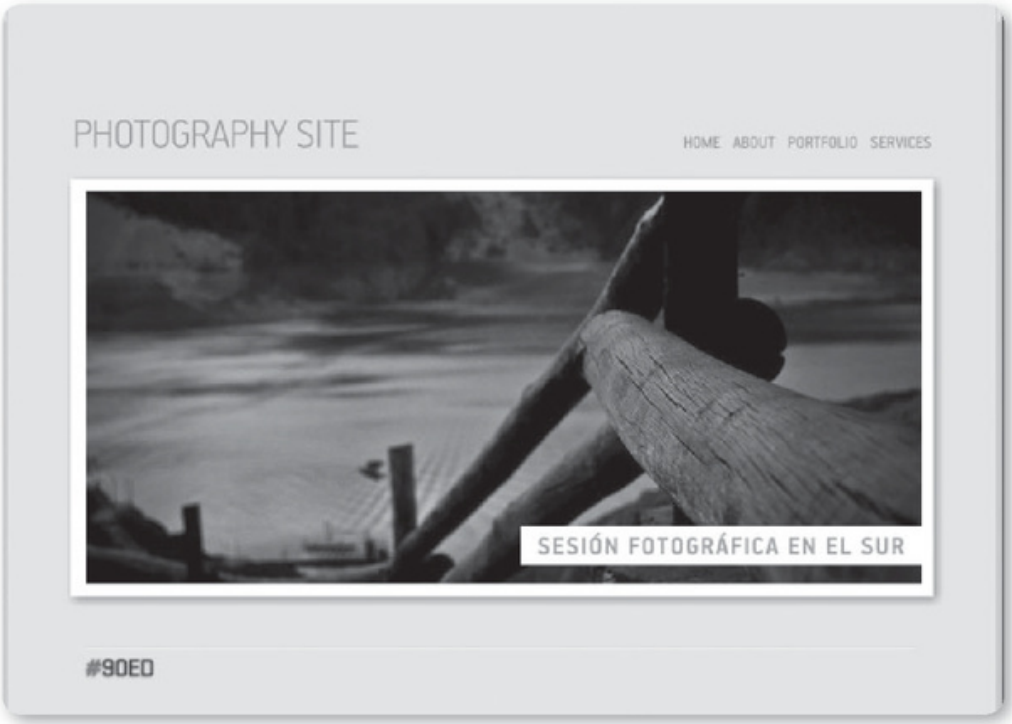



**Figura 1.** El sitio que desarrollaremos como ejemplo nos permitirá explorar todas las funcionalidades de nuestro framework.

## MODELO DEL FRAMEWORK Y DEL SITIO

El modelo del espacio de trabajo es parte del MVC que conforma la estructura básica y hace funcionar nuestro sistema, mientras que el modelo del proyecto tiene funcionalidades específicas para el proyecto en sí. No debemos confundir ambos conceptos.

# Archivos .FLA

Lo primero que debemos hacer para iniciar el desarrollo de nuestro sitio es crear los archivos **.FLA** correspondientes a cada una de las secciones que tendrá (uno por cada nodo **page** que contiene el archivo **site.xml**). De esta manera, dentro de nuestra carpeta **lib** deberíamos de tener listos los archivos **index.fla**, **about.fla**, **services.fla**, **preloader.fla**, **home.fla** y **portfolio.fla**. Es importante recordar que debemos definir como destino de publicación la carpeta **swf** situada dentro de **bin**, a excepción del archivo **index.fla**, que deberá ser exportado dentro de **bin** porque el mismo **.SWF** se ocupará de cargar el resto de las películas. Quien no recuerde el procedimiento puede consultar el **Capítulo 3**, donde explicamos cómo exportar contenido a otro directorio. Dentro de la estructura que hemos definido para nuestro entorno de trabajo (carpeta bin, carpeta src, carpeta lib, etcétera).

> LO PRIMERO QUE DEBEMOS HACER ES REALIZAR LA CREACIÓN DE LOS ARCHIVOS .FLA

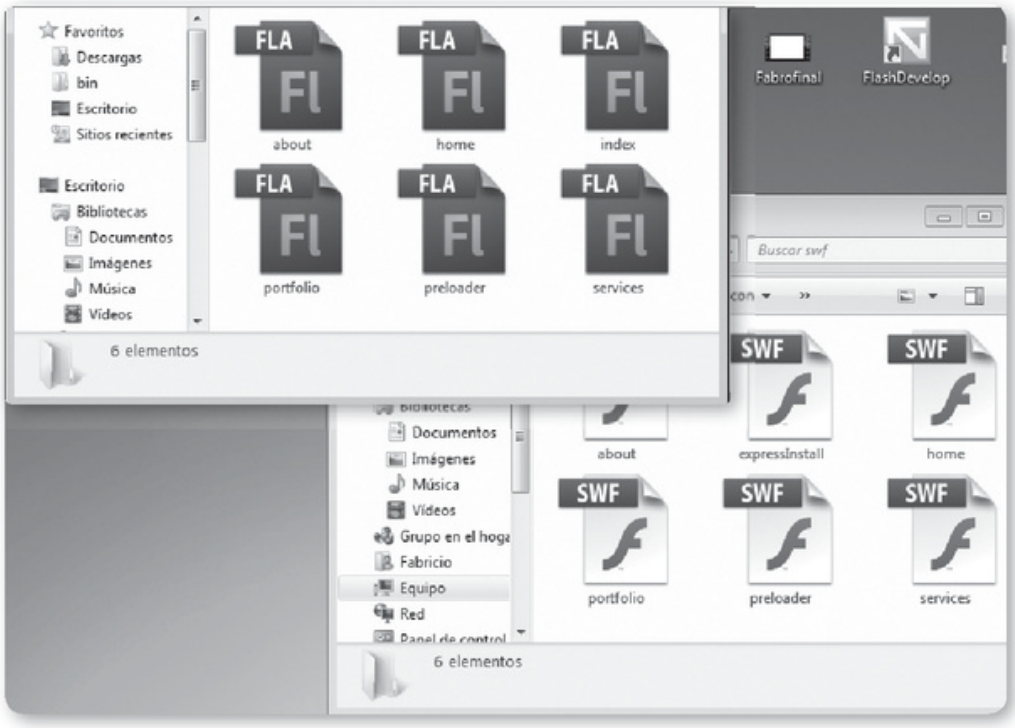



**Figura 2.** Los archivos **.FLA** y **.SWF** del proyecto, en sus respectivas carpetas. Mantenemos separado el diseño de la implementación.

# Paquete project

Como mencionamos en el **Capítulo 3**, el paquete **project** contiene todas las clases que hacen al funcionamiento de un proyecto en particular. Dentro de **project**, crearemos los siguientes paquetes:

- **model**: contiene las clases que manejen datos de nuestras películas.
- **display**: almacena las clases que manejen elementos gráficos.
- **pages**: es uno de los paquetes más importantes, ya que incluye las clases que dan inicio y funcionalidad a cada uno de nuestros archivos fuente (los **.FLA** de sección). Más adelante en este capítulo, veremos cómo crear cada una de estas clases que pertenecen al paquete **pages** y explicaremos su importancia.

Es fundamental no confundir el modelo del *framerwork* con el modelo propio de nuestro proyecto.

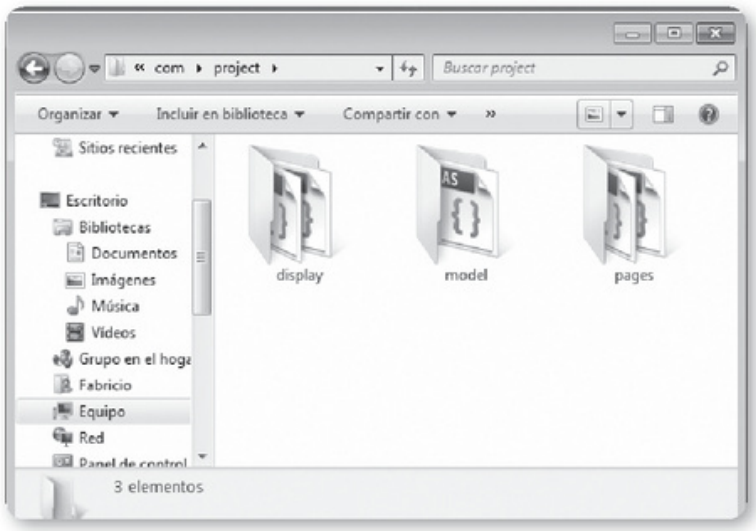



**Figura 3.** Así como separamos el diseño de la implementación, también separamos el código del framework, del proyecto.

## PROPIEDAD MASK

Para asignar máscaras a los contenidos, debemos hacer uso de la propiedad **mask**. El **displayObject** 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**. Es importante que tengamos en cuenta que podemos asignar o eliminar las máscaras cuantas veces sean necesarias.

# Inicializar el framework

A la hora de querer emplear nuestro espacio de trabajo, debemos inicializarlo para poder hacer uso de todo su potencial. A continuación veremos cómo, mediante unas sencillas líneas de código, podemos comenzar a hacer uso de los recursos de nuestro *framework*. Pese a la sencillez de este proceso, debemos familiarizarnos con él: cada vez que comencemos un nuevo proyecto, tendremos que extender su clase principal (generalmente, llamada **Main**) a la clase principal de nuestro *framework* (nosotros la hemos denominado **FrameworkMain**). Una vez que nuestro espacio de trabajo nos avisa que está listo para utilizarse, ya podemos comenzar a crear la interfaz gráfica de usuario. A continuación, veremos de qué manera hacerlo.

## Main.as

Es importante tener en cuenta que siempre que nos embarquemos en la tarea de crear un nuevo proyecto, debemos comenzar por vincular la clase **Main.as** al archivo fuente principal que contendrá los elementos de navegación y, a su vez, se ocupará de cargar las páginas; de esta forma podremos iniciar la programacion (en nuestro caso, **index.fla**).

> LA PROGRAMACIÓN ORIENTADA A OBJETOS NOS BRINDA LA POSIBILIDAD DE EXTENDER CLASES

Una de las ventajas que nos brinda la Programación Orientada a Objetos es la posibilidad de **extender clases**. Esto nos permite mantener una estructura interna de código idéntica para todos nuestros proyectos, pero, a la vez, nos da la libertad de desarrollar distintas funcionalidades para cada caso en particular. Este es el caso de la clase **FrameworkMain.as**, la cual vamos a extender en todos los proyectos que utilicen el entorno de trabajo desarrollado en capítulos anteriores.

Si miramos el código de la clase **Main.as**, veremos que hay dos funciones que poseen el prefijo **override protected**, el cual es utilizado para sobrescribir funciones de clases extendidas: extender a **FrameworkMain** nos permite contar con los beneficios de nuestro espacio de trabajo, y trabajar las partes específicas desde **Main**, sin mezclar el código que corresponde a cada entorno (el propio y el *framework*).

```
public class Main extends FrameworkMain
    {
        public var menu:Menu;
        private var footer:Footer;
        private var title:Title;
        private var swfTracker:SWFTracker;


        public function Main()
        {
            super();
            siteXML = "xml/site.xml";
        }

        override protected function onAddedToStage(event:Event):void
        {
            super.onAddedToStage(event);
            stage.align = StageAlign.TOP_LEFT;
            stage.scaleMode = StageScaleMode.NO_SCALE;

this.x = (stage.stageWidth - Constants.CONTENT_WIDTH) / 2;
            this.y = (stage.stageHeight - Constants.CONTENT_HEIGHT) / 2;

        }

        override protected function init():void {
            super.init();

          Menu.getInstance().init();
```

## TAMAÑO MÍNIMO Y MÁXIMO PARA UN SITIO

Es importante saber que una buena técnica para el desarrollo de *Liquid Layouts* es definir un tamaño mínimo y máximo de ancho y largo para los sitios. De esta manera, tendremos un mayor control al momento de mostrar el contenido y crear un mejor diseño de la interfaz.

```
            addChild(Menu.getInstance());

            title = new Title();
            title.addEventListener(MouseEvent.CLICK, onClickTitle, false, 0, true);
            title.buttonMode = true;
            addChild(title);

            footer = new Footer();
            footer.x = 16;
            footer.y = 600;
            addChild(footer);
        }
```

En la función **onAddedToStage()** debemos hacer un llamado a la clase extendida a través de **super()**, y le pasamos como parámetro el evento recibido al haber sido añadido en el escenario. Así, la clase **FrameworkMain** puede dar inicio al resto de las clases necesarias para el *framework*. Dentro de esa función, podemos añadir las líneas de código que consideremos necesarias antes de que se inicialice la aplicación. Para este ejemplo, simplemente situaremos la película en el centro del escenario.

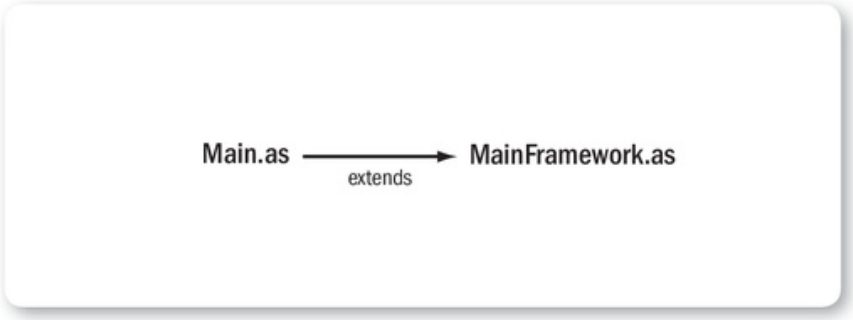



**Figura 4.** Al extender la clase **`FrameworkMain`**, podremos sobrescribir sus funciones principales para dar comportamientos específicos a nuestra clase.

Otra de las funciones que vamos a extender es **init()**, que se ejecuta una vez que el *framework* se inicia de manera exitosa.

# Liquid Layout

Hemos hecho una pequeña introducción al tema en el **Capítulo 7** de este libro al hablar sobre dispositivos, pero por el momento, centrémonos en la Web, que es donde mejor se adapta nuestro espacio de trabajo: cuando desarrollamos un sitio, muchas veces precisamos que los elementos de la interfaz gráfica se acomoden en una determinada posición dependiendo de las medidas de la ventana del navegador en el cual se ha cargado nuestra película. Sin embargo, es posible que el usuario redimensione la ventana y, entonces, se rompa la disposición original. Para evitar este problema, podemos emplear el evento **RESIZE**: ya lo hemos mencionado en el capítulo destinado a diseño para dispositivos:

```
stage.addEventListener(Event.RESIZE, onResize);
```

En nuestro caso, vamos a utilizar este evento para centrar la película.

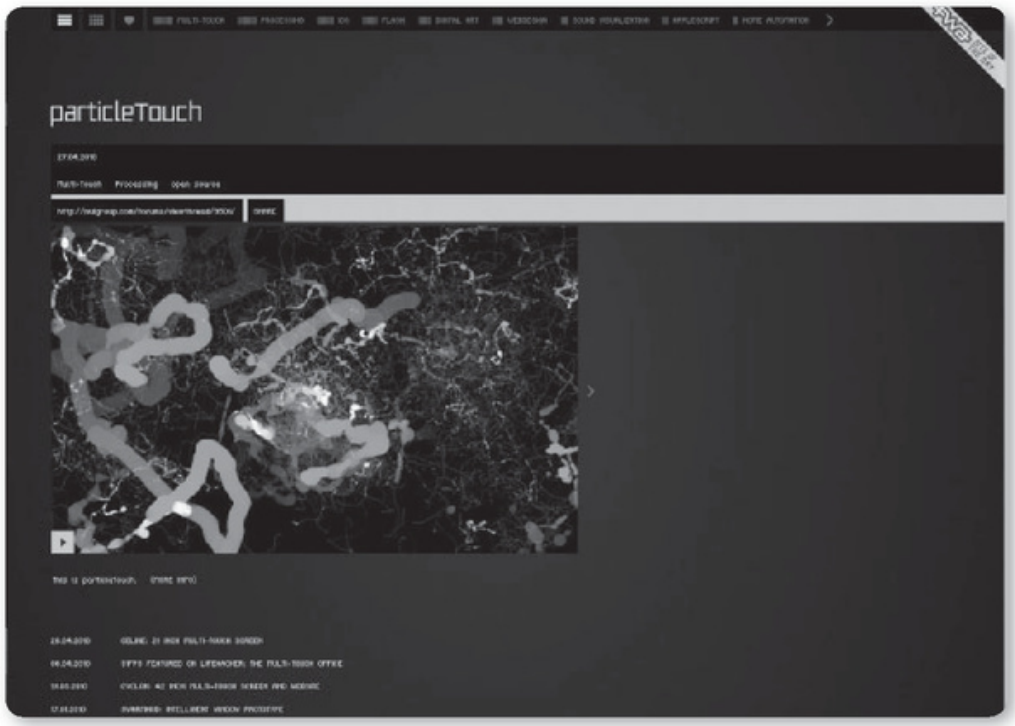



**Figura 5.** El sitio **www.thirtyonefps.com/flash.php** es un excelente ejemplo para entender el concepto de Liquid Layout.

Cada vez que se redimensione la ventana del navegador, se ejecutará la función **onResize**. Si bien podríamos generar distintas disposiciones dependiendo del tamaño del escenario, para este caso, simplemente, vamos a situar la película en el centro.

Es muy importante que tengamos en cuenta que en este caso particular, la película se situará en el centro del navegador, y por consiguiente nos encargaremos de mantener siempre este posicionamiento para la reproducción, independientemente de las dimensiones que tome el mismo.

```
private function onResize(e:Event):void
{
    this.x = (stage.stageWidth - Constants.CONTENT_WIDTH) / 2;
    this.y = (stage.stageHeight - Constants.CONTENT_HEIGHT) / 2;
}
```


**Figura 6.** Independientemente del tamaño de la ventana, el contenido del sitio siempre aparecerá centrado.

# Preparar el contenido de cada sección

Una vez que el *framework* se ha inicializado correctamente y hemos centrado el contenido, nos resta generar el contenido que vamos a mostrar en nuestro sitio. Para hacerlo, el primer paso será generar una clase que vincularemos con cada uno de los archivos fuente que pertenezcan a una sección del sitio.

Recordemos cuáles son las secciones que creamos: **about.fla**, **services.fla**, **home.fla** y **portfolio.fla**. Teniendo en cuenta esto, necesitamos crear las clases para cada una de las secciones: **AboutPage.as**, **ServicesPage.as**, **HomePage.as** y **PortfolioPage.as**, las cuales situaremos en el paquete **com.project.pages**. Cada una de las clases que mencionamos anteriormente extenderá a la clase **AbstractSection**, y **AbstracSection** extenderá a **Sprite**. En las próximas páginas explicaremos la clase **AbstracSection** y la gran ventaja de emplearla.

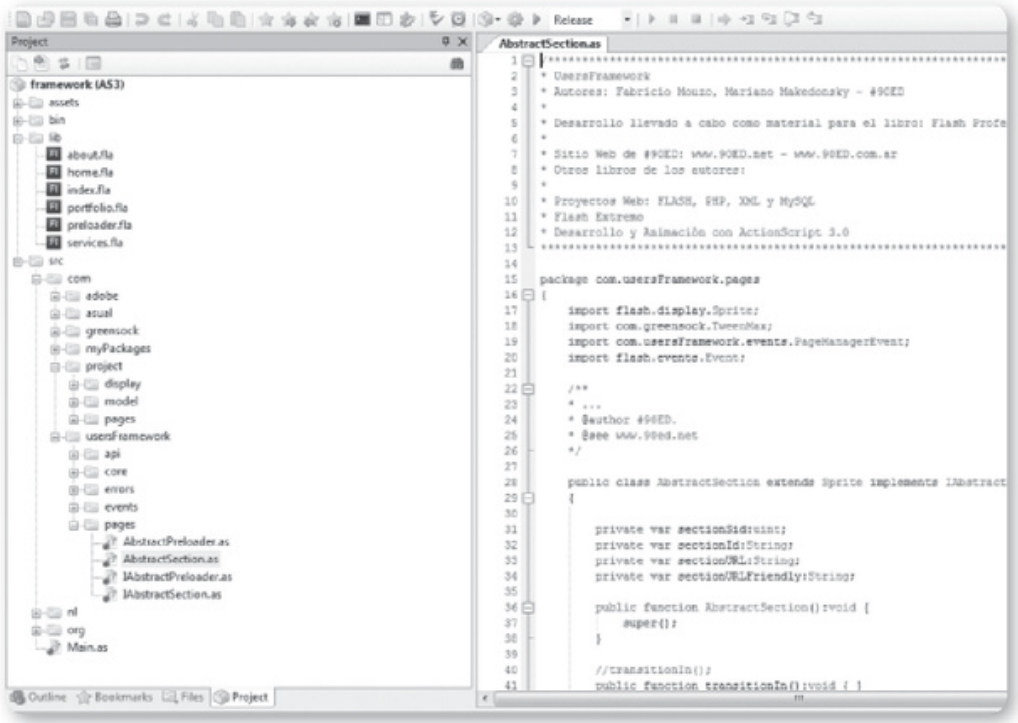



**Figura 7.** Las clases vinculadas a cada archivo de sección se almacenan en el paquete **com.project.pages**.

# HomePage.as

A fin de no ser reiterativos con las explicaciones, solamente veremos cómo darles forma a las clases **HomePage.as** y **PortfolioPage.as**. El proceso que se explica a continuación nos servirá para todas las secciones que deseemos integrar al proyecto: simplemente, hay que repetir los pasos.

## Extender a AbstractSection

Insistimos interminables veces a lo largo de este libro sobre la importancia de pensar en la optimización de nuestro desarrollo y la reutilización. Pensar de esta manera nos ayuda a encontrar soluciones a varios problemas de índole similar. Eso mismo es lo que hace la clase **AbstractSection**: se encarga de que podamos tener, en una clase, funcionalidades que precisaremos en muchas películas.

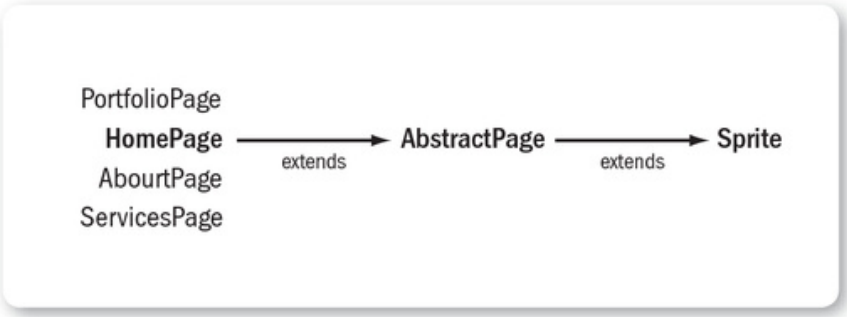



**Figura 8.** Cada clase que se vincula a una sección del sitio debe extender a la clase **AbstractPage**, y **AbstractPage** extiende a **Sprite**.

Como dijimos, cada página debe extender a **AbstractSction**: esto implica que los métodos y las propiedades de la clase **AbstractSection** pasarán a formar parte de la clase principal de cada página (en este caso, **HomePage**). Esta clase contiene cuatro métodos que son imprescindibles para el funcionamiento del sitio:

```
public function transitionIn():void
public function transitionOut():void
```

```
public function transitionInCompleted():void
public function transitionOutCompleted():void
```

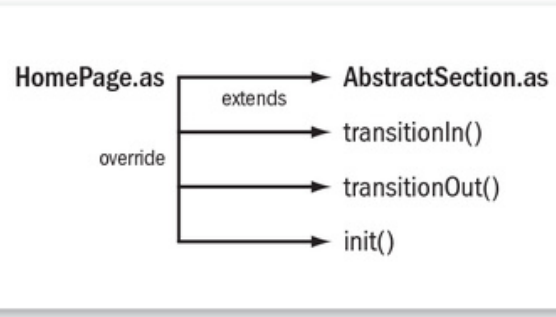



**Figura 9.** En esta imagen vemos que la clase **AbstractSection** contiene los métodos y eventos con los que se comunica el controlador del sitio.

Si abrimos el controlador del sitio (**FrameworkController**) y recordamos lo que hemos repasado en capítulos anteriores, sabremos que, desde esa clase, manejamos la carga y descarga de contenidos. Veamos el método **setTransitonIn()** del controlador:

```
private function setTransitionIn():void {
    frameworkModel.addPage(pendingPage);
    pendingPage.addEventListener(PageManagerEvent.TRANSITION_IN_
        COMPLETED, transitionInCompleted);
    pendingPage.transitionIn();
}
```

Son tres sencillos pasos: la primera línea le pide al modelo que añada la página al escenario, la segunda asigna un *listener* para saber cuándo se completó la transición de entrada, y la tercera ejecuta el método **transitionIn()** de la página actual.

Ahora bien, ¿cuál es el método **transitionIn()** de cada página? Nada más y nada menos que el método que contiene la clase **AbstractSection**:

```
public function transitionIn():void {
    TweenMax.to(this, 0.3, { alpha:1, onComplete:transitionInCompleted } );
}
```

¿Y desde dónde se ejecuta el *listener* para informarle al controlador que se completó la transición de entrada? También desde **AbstractSection**, veremos que es una clase sumamente útil y necesaria. Encontramos un claro ejemplo en el siguiente código:

```
public function transitionInCompleted():void {
    dispatchEvent(new Event(PageManagerEvent.TRANSITION_IN_
        COMPLETED));
}
```

## Nexo entre el framework y el proyecto

La clase **AbstractSection** recibe información y da información al controlador del sitio. Cuando se inicia una nueva carga, el controlador le dice a cada página actual que ejecute el método **transitionIn()**. Cuando se completó la transición de entrada, **AbstractSection** se lo informa al controlador ejecutando el evento **TRANSITION_IN_IS_COMPLETED**. Cuando se inicia una descarga, el controlador le indica a **AbstractSection** que ejecute el método **transitionOut()**. Cuando se completó la transición de salida, **AbstractSection** se lo informa al controlador por medio del evento **TRANSITION_OUT_IS_COMPLETED**.

### VARIAS SECCIONES

La ventaja de haber pensado nuestro espacio de trabajo de esta manera es que, con tres sencillos pasos, podemos agregar una nueva sección al sitio y lograr que esta sea funcional dentro del entorno: podremos acceder a ella desde el menú, desde el menú contextual, desde la URL o por medio de un atajo de teclado sin que necesitemos programar nada. El ahorro de tiempo que logramos con esto es inmenso y realmente significante, por esta razón lo debemos tener en cuenta para nuestros desarrollos.

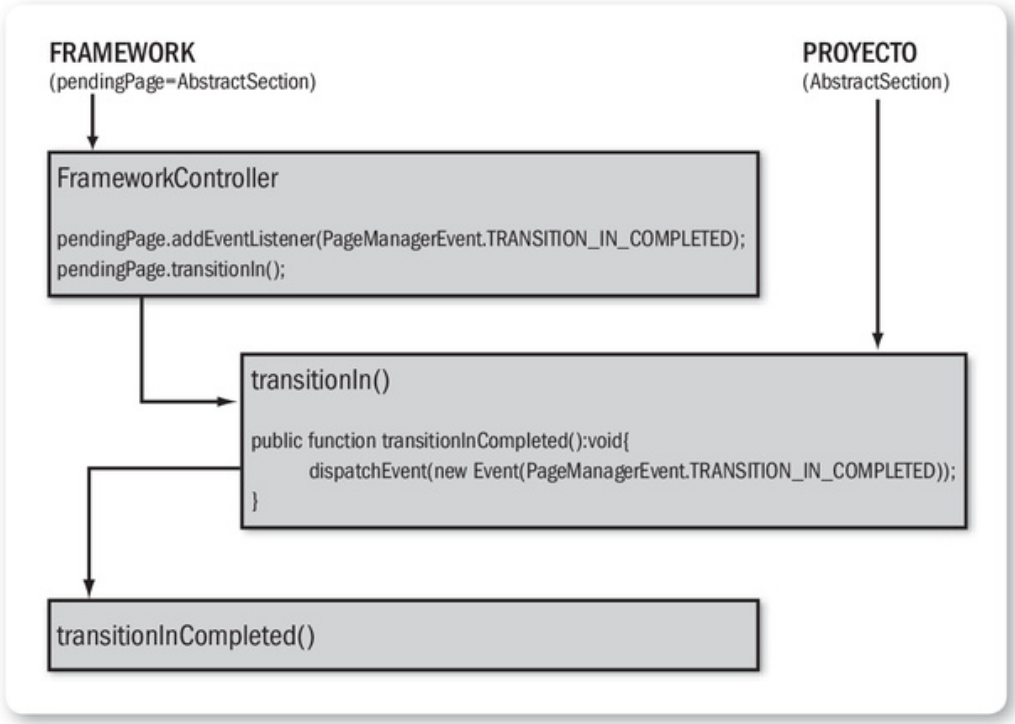



**Figura 10.** Comunicación entre el framework y nuestro proyecto al realizar un proceso de carga de sección.

El hecho de trabajar las secciones de esta manera nos da una inmensa ventaja: todas las funciones que se comunican con el controlador son manejadas por una única clase. De no hacerlo de este modo, tendríamos que rescribir código constantemente en cada una de las secciones. Ahora bien, podríamos pensar que esto nos plantea el siguiente problema: al tener las transiciones de entrada y salida del sitio en una única clase, no podemos personalizar las transiciones para que sean distintas en cada sección. En caso de que esto sea necesario,

## NUEVAS FUNCIONALIDADES

Es necesario tener en cuenta que lejos de considerar nuestro espacio de trabajo como una solución final, sería ideal que cada desarrollador implementara las partes que realmente le resultaran útiles e interesantes de él y las adaptara a su propio entorno, o bien, tomara este desarrollo como punto de partida y le agregara nuevas funcionalidades: siempre hay algo por mejorar.

hacemos uso del método **override** desde la clase principal, en este ejemplo, **HomePage**, tal como vemos a continuación:

```
override public function transitionIn():void {
    TweenMax.to(this, 0.3, { alpha:1, onComplete:transitionInCompleted } );
}
override public function transitionOut():void {
    TweenMax.to(this, 0.3, { alpha:0, onComplete:transitionOutCompleted } );
}
```

Por medio de **override**, lo que hacemos es decirle a nuestro código que reemplace la función que lleva su mismo nombre. Al hacer un **override** de las funciones **transitionIn()** y **transitionOut()**, le estamos indicando que ejecute funciones con ese nombre que se encuentran en la clase principal, y no, las que están dentro de **AbstractSection** (al ser una clase que extiende a la principal, sus métodos son parte de ella misma). De esta manera, si queremos personalizar la entrada y la salida de cada sección, lo hacemos desde la página principal de cada sección. Lo importante es que, una vez concluidas las transiciones, se ejecuten los métodos **transitionOutCompleted** y **transitionInCompleted**, ya que estas funciones son las que le avisan al controlador que se completaron las transiciones.

## Iniciar la sección actual

De todos los métodos que contiene la clase **AbstractSection** hay uno solo que tenemos la obligación de reemplazar (**override**): el método **init()**. Si prestamos atención, veremos que cuando se ejecuta el método **transitionInCompleted()** del controlador, se le indica a la página actual que ejecute el método **init()**:

```
private function transitionInCompleted(e:Event):void {
pendingPage.removeEventListener(PageManagerEvent.TRANSITION_IN_
    COMPLETED, transitionInCompleted);
    pendingPage.init();
    actualPage = pendingPage;
}
```

La función **init()** es la que le da inicio a todo lo que sucede dentro de la sección que acabamos de cargar.

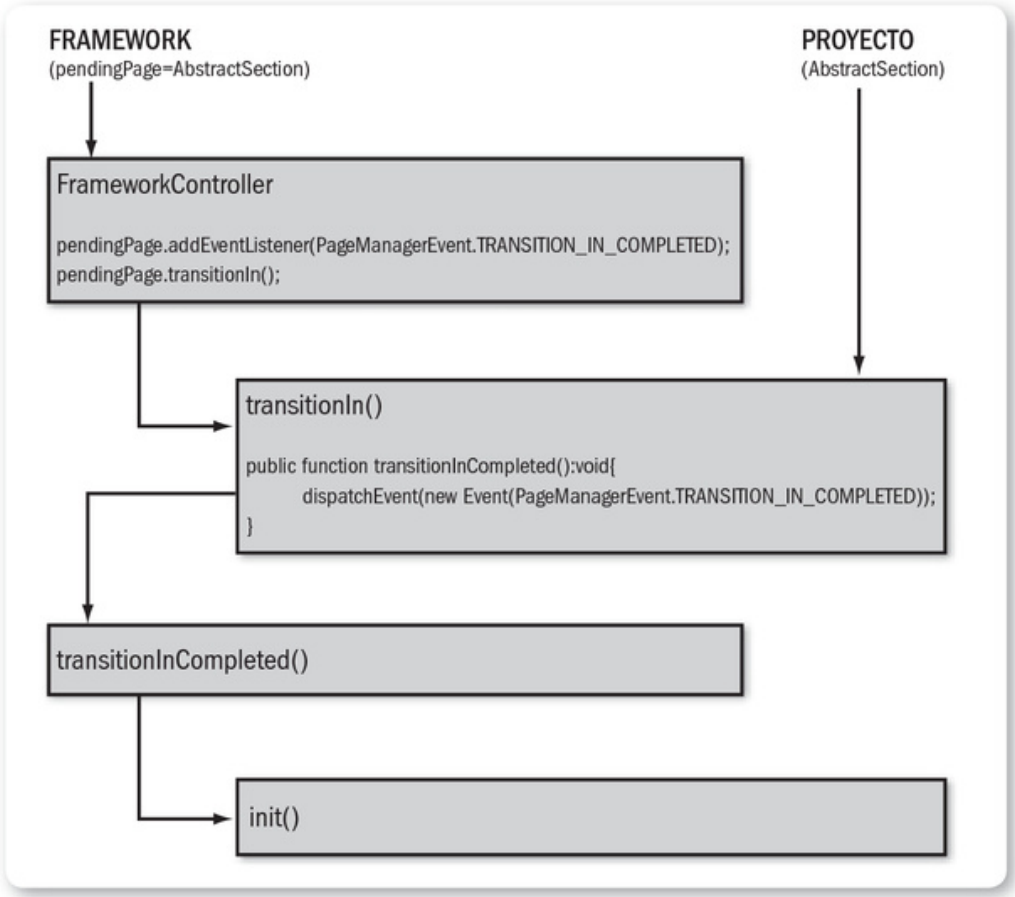



**Figura 11.** Desde el framework se llama al método **`init()`** de la sección cargada. Este método se ejecuta una vez que hemos cargado la sección actual: es el punto de inicio.

## Otros métodos de la clase AbstractSection

Si bien el método **init()** es el único que tenemos que reemplazar para inicializar cada sección, anteriormente dijimos que se puede reemplazar cualquier método de la clase **AbstractSection**: hemos dado el ejemplo de los métodos **transitionIn()** y **transitionOut()**, pero también hay otros métodos que podemos llegar a necesitar en cualquier proyecto:

**setParams()** recibe los parámetros especiales de los cuales hablaremos más adelante, **setURL()** contiene la URL de la sección

cargada (**../home.swf**), **setSid()** obtiene el identificador numérico de la sección, **setId()** es el id propio de la página, y **setURLFriendly()** es el identificador para emplear al hacer uso de **SWFAddress**. Si bien no usaremos ninguno, es importante saber que la clase abstracta **AbstractSection** cuenta con ellos.

**Figura 12.** Es importante comprender con claridad cuál es la función de la clase **AbstractSection**, los métodos y para qué sirven.

# La razón de ser de nuestro framework

No olvidemos que la explicación que hemos hecho anteriormente no es una más de este libro: es, indudablemente, la más importante de todas, y sobre la que recae todo nuestro esfuerzo y trabajo previo; por esta razón siempre debemos tenerla en cuenta. Este tipo de desarrollo implica un sinfín de clases, de técnicas, de mucha Programación Orientada a Objetos, de patrones, de criterios de usabilidad y accesibilidad y de SEO, y seguramente nos preguntemos, ¿para qué?

El método **init()** de cada sección es la explicación a esto. En el **Capítulo 2** de este libro entregamos una definición de espacio de trabajo que volveremos a repetir en esta ocasión.

La gran ventaja de un espacio de trabajo radica en la posibilidad de agrupar aquellos elementos en común que vamos a utilizar en todos (o

casi todos) nuestros desarrollos, para poder hacer uso de ellos cuando la situación lo requiera, sin tener que programarlos nuevamente cada vez que surja uno nuevo.

A su vez, ¿qué nos permite contar con un espacio de trabajo? También lo dijimos en el mismo capítulo:

Una vez que contamos con un entorno de desarrollo organizado, podemos utilizarlo como punto de partida para cualquier proyecto que hagamos, y así, optimizaremos nuestros **tiempos** y **recursos**. De esta manera, podemos abocarnos de lleno a la parte más importante de los proyectos, que son las especificidades de cada desarrollo y las características particulares de cada uno de ellos. Veamos los beneficios.

## Abocarnos a lo específico de un desarrollo

El método **init()** de cada sección es el punto de partida de esas especificidades del desarrollo de las que hablamos en las primeras páginas: al haber hecho este espacio de trabajo y extender nuestros proyectos a él, tenemos todas sus funcionalidades ya programadas y podemos centrarnos en lo que realmente nos interesa y donde tenemos que poner nuestros esfuerzos.

## Entorno optimizado para grupos interdisciplinarios

Como consecuencia de esta metodología de desarrollo, nos queda conformado un entorno dentro del cual varias personas pueden trabajar en un mismo proyecto, sin necesidad de "pisarse" durante el desarrollo y con tareas bien específicas: sabemos que, extendiendo a **AbstractSection** y haciendo un **override** del método **init()**, tendremos nuestra película integrada con el *framework*.

### SVN PARA DESARROLLO

Muchas veces necesitamos trabajar en equipo en un mismo proyecto. La mejor solución en estos casos es implementar un servidor SVN para poder tener un historial de las versiones, llevar un mayor control sobre el desarrollo y conseguir una mejor interacción entre todos los desarrolladores.

```
*
* Titan Framework es un espacio de trabajo creado para utilizar con Adobe Flash.
* Titan Framework es liberado bajo la licencia GPL.
*
* Para más información sobre GPL:
* http://www.opensource.org/licenses/gpl-2.0.php
*****************************************************************************************/

package com.project.pages
{
    import com.greensock.TweenMax;
    import com.usersframework.api.Framework;
    import com.usersframework.pages.AbstractSection

    /**
    * ...
    * @author #90ED.
    * @see www.90ed.net/titanFramework
    */

    public class ServicesPage extends AbstractSection

    {
        public function ServicesPage():void { this.alpha = 0;
            this.x = -960;
        }
        override public function transitionIn():void {
            TweenMax.to(this, 0.3, { alpha:1, x: 0, onComplete:transitionInCompleted } );
        }
        override public function transitionOut():void { TweenMax.to(this, 0.3, { alpha:0, x:(stage.stageWidth + 960),
        override public function setParams(params:Array):void { }
        override public function init():void {
        }

    }
}
```

**Figura 13.** Para que una página pase a utilizar las funcionalidades del framework, debemos extenderla a **`AbstractSection`**.

# Carga y descarga de contenido: preloader

Si bien muchas veces no les damos la importancia necesaria, los *preloaders* nos permiten mantener informados a los usuarios sobre el estado de nuestra aplicación durante un proceso de carga.

El *framework* que hemos desarrollado contempla esta situación: del mismo modo en que creamos la clase **AbstractSection** para las secciones del sitio, también hemos creado la clase **AbstractPreloader**. Su funcionamiento es muy similar al de **AbstractSection**: la utilizamos para extender a ella nuestro propio *preloader*.

LOS PRELOADERS NOS PERMITEN MANTENER INFORMADOS A LOS USUARIOS

A los fines prácticos, el *preloader* que utilizaremos en este sitio es muy sencillo: se trata, simplemente, de una delgada barra que se escalará hasta completar el ancho del escenario anunciando la carga de una nueva sección. Pero veremos que, independientemente de su visualización, tendremos resuelto el código para tener la información necesaria dentro de la película. La creatividad para luego hacer algo interesante con esa información recae en cada desarrollador.

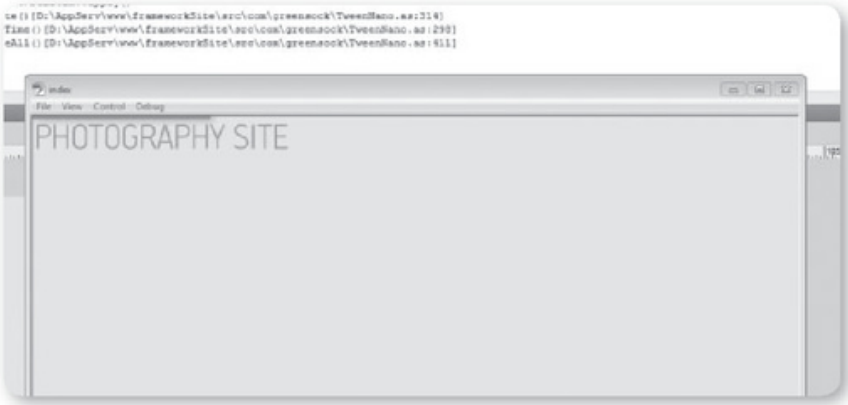



**Figura 14.** Nuestro preloader, en funcionamiento: utilizamos el largo del escenario para que se escale en tu totalidad.

# Pensar en nuestro preloader

Debemos nombrar nuestro *preloader* como **preloader.fla** y colocarlo en la carpeta **libs**. A diferencia del resto de los **.FLA** que vinculamos a su clase de sección, el *preloader* lo vinculamos a la clase **FrameworkPreloader.as**, la cual se encuentra en el paquete **com.project.pages**.

**Figura 15.** Es posible encontrar el clip que utilizamos como preloader en la librería del archivo **`preloader.fla`**.

Lo que nos interesa de esta clase es que extiende a la clase **AbstractPreloader**. Si abrimos esta última (**com.usersFramework.pages**), encontraremos una estructura muy similar a la de **AbstractSection**:

```
public class AbstractPreloader extends Sprite implements IAbstractPreloader
    {
    public function AbstractSection():void {}

    //transitionIn();
    public function transitionIn():void { }

    //transitionOut();
    public function transitionOut():void{ }

    //transitionInCompleted();
    public function transitionInCompleted():void {
        dispatchEvent(new Event(PageManagerEvent.TRANSITION_IN_
            COMPLETED));
    }
    //transitionOutCompleted();
    public function transitionOutCompleted():void {
        dispatchEvent(new Event(PageManagerEvent.TRANSITION_OUT_
            COMPLETED));
}
//progreso!
    public function setProgress(bytesLoaded:Number, bytesTotal:Number):void {
            }
```

Su funcionamiento es exactamente igual al de cada página de sección, a diferencia de que, desde **FrameworkPreloader**, no tenemos la necesidad de hacer ningún **override** de ningún método de **AbstractPreloader**. Lo hacemos únicamente en función del tipo de *preloader* que queremos generar:

## AÑADIR SECCIONES AL SITIO

Recordemos que utilizando el sistema planteado a lo largo del capítulo, es muy fácil escalar sus dimensiones. Si necesitamos agregar una nueva sección, basta con añadirla al **site.xml** y, posteriormente, crear su correspondiente archivo fuente con su respectiva clase.

```
public class FrameworkPreloader extends AbstractPreloader
{

    private var bytesLoaded:Number;
    private var bytesTotal:Number;
    public var barra:MovieClip;

    public function FrameworkPreloader():void {
        this.alpha = 1;
        this.addEventListener(Event.ADDED_TO_STAGE, onAdded)
    }
private function onAdded(event : Event) : void {
        barra.y = stage.stageHeight + 10;
        barra.width = stage.stageWidth;
        this.scaleX =0;
    }

override public function transitionIn():void
{
    var posY:uint = 0;
    trace(stage.stageWidth);
    TweenMax.to(this,0.3,{alpha: 1});
    TweenMax.to(barra, 0.3, {alpha:1, y: posY, onComplete:transitionInComplete
        d});
}
override public function transitionOut():void
{
    TweenMax.to(this, 0.3, { alpha:0, onComplete:transitionOutCompleted } );
}

override public function setProgress(bytesLoaded:Number,
    bytesTotal:Number):void {
        this.scaleX = bytesLoaded / bytesTotal;
}
```

En este caso, lo que hicimos fue sobrescribir las funciones de animación de entrada y salida, y otra función llamada **setProgress()**:

```
public function setProgress(bytesLoaded:Number, bytesTotal:Number):void {
    this.scaleX = bytesLoaded / bytesTotal;
}
```

La función **setProgress()** se ejecutará cada vez que se actualicen los bytes cargados. En nuestro caso, lo que hacemos es utilizar esta información para modificar el ancho de la barra de precarga, pero independientemente de esto, lo importante es que, contando con esta información dentro de la película, podemos generar el tipo de animación que creamos necesario. Con esta sencilla línea dentro del método **setProgress()**, podríamos indicarle la precarga a un campo de texto:

```
porcentaje_txt.text = Math.round((bytesLoaded / bytesTotal) * 100).toString();
```

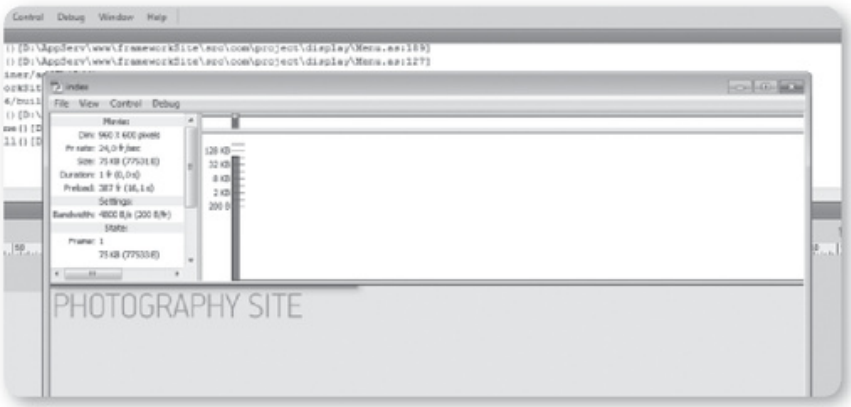



**Figura 16.** Al presionar **CTRL+B** una vez exportada la película, accedemos al menú **Visor de anchos de banda**.

# Pensar los preloaders de manera usable

Los tiempos cambian, y con ellos, nuestras costumbres y exigencias sobre la Web. Lamentablemente, este es un libro de programación, y desde el momento en el que le brindamos a cada lector una clase que contiene toda la información necesaria para crear un *preloader* y un

ejemplo de eso, sabemos que cumplimos con nuestro objetivo. Pero la multimedia va mucho más allá. A lo largo del **Capítulo 6** nos referimos en varias oportunidades a la importancia de brindar una buena experiencia. Una barra de progreso o un campo de texto es una buena experiencia (o, tal vez, una experiencia imperceptible) si la carga es rápida, pero cuando los contenidos por cargar llevan su tiempo, debemos ir más allá y pensar de manera creativa. Hay miles de formas de ofrecerle a un usuario un mejor momento en nuestra web o aplicación cuando los tiempos de carga son considerablemente altos: desde la manera más sencilla de interacción hasta un juego pueden formar parte de un proceso de carga. Lo importante es pensar con creatividad, y es aquí donde se hace fuerte nuestro *framework* y este libro: no orientamos nuestros esfuerzos a generar un entorno que no nos haga trabajar por el resto del desarrollo: lo que buscamos es no reinventar constantemente la rueda, a fin de obtener tiempo y poder invertirlo en lo específico de cada desarrollo. Los bytes cargados y los bytes totales dependen de Flash. El hecho de hacer algo interesante con ellos depende de nosotros.

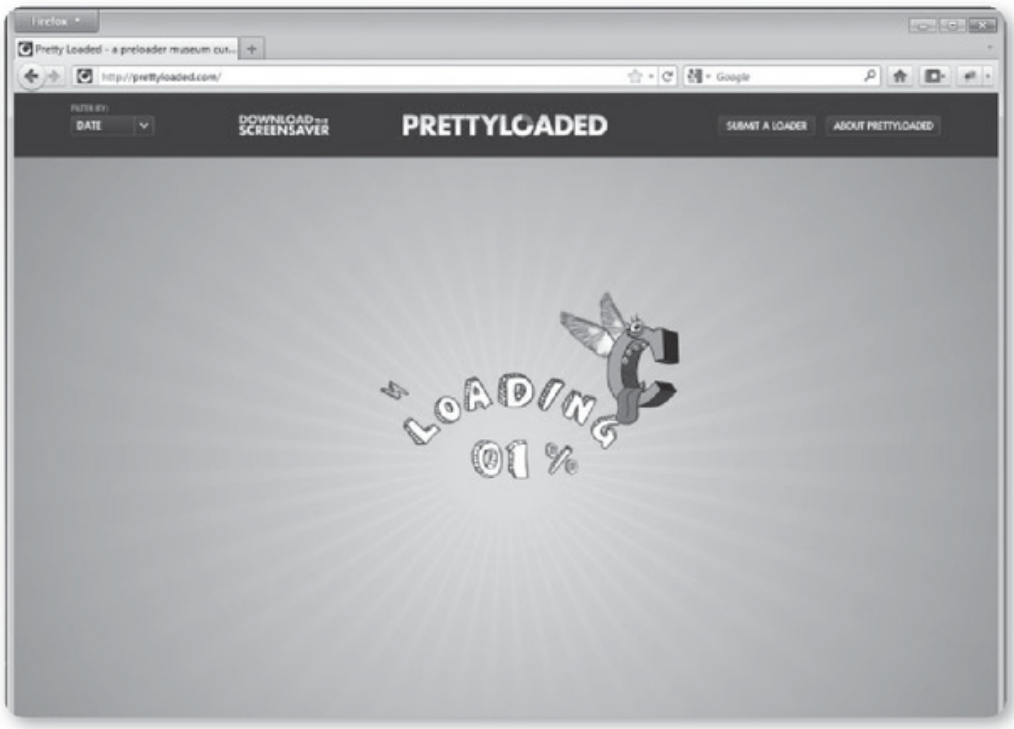



**Figura 17.** En **`www.prettyloaded.com`** podemos encontrar una enorme galería de preloaders para nuestra inspiración.

# Galería de imágenes con deep linking

La sección **Portfolio** de nuestro sitio posee ciertas peculiaridades que nos obligan a explicarla con mayor detenimiento.

Si miramos el sitio de ejemplo, veremos que esta sección se ocupa de generar una galería de imágenes dinámica en la cual, cada vez que realizamos un clic en alguna de las miniaturas (**thumbnails**), la imagen seleccionada se carga en tamaño grande. Lo que resulta interesante de este ejemplo sucede en la barra de navegación de nuestro navegador: veremos que, al seleccionar una imagen, cambia la URL, pero a diferencia de lo que hemos visto en el capítulo sobre usabilidad, hay más de un parámetro. Este representa un enorme beneficio en los más diversos modos: afecta de forma positiva la usabilidad y la accesibilidad e, incluso, podemos hacer que sea útil para SEO. A continuación, veremos de qué manera manipular estos parámetros adicionales de la URL.

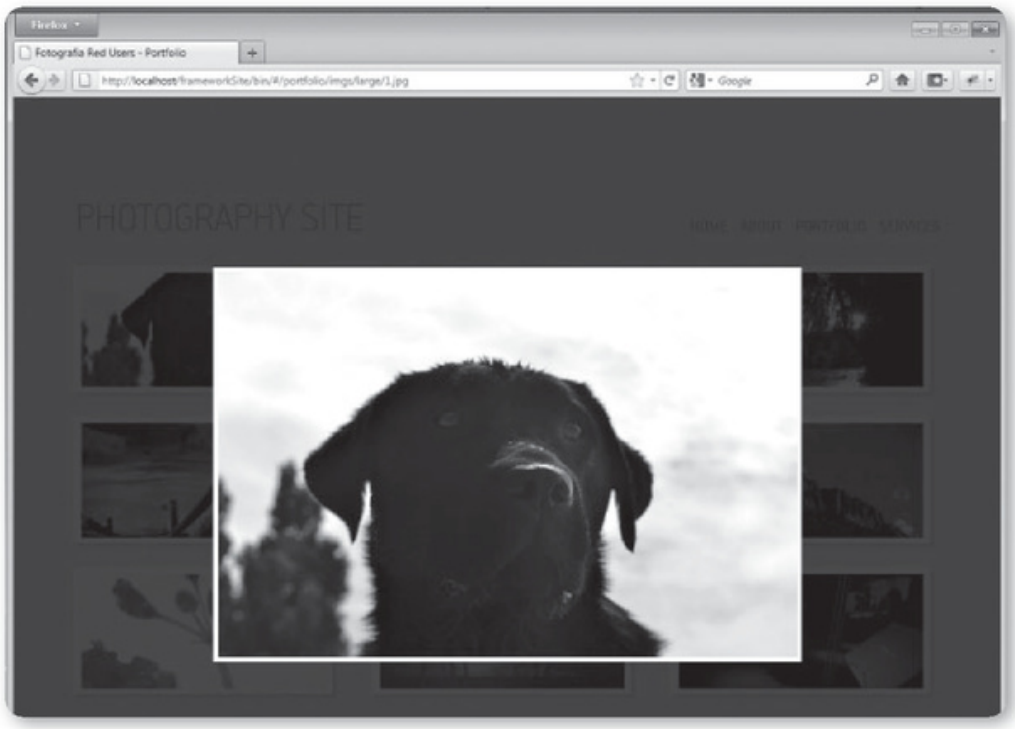



**Figura 18.** Aquí vemos nuestra galería de imágenes en pleno funcionamiento. Prestemos atención a la barra del navegador.

# Volver a la carga sobre usabilidad

En general sucede que los sitios en Flash no trabajan el contenido de las secciones en profundidad (*deep linking*), y si visitamos uno, independientemente de la sección en la que nos encontramos, no cambia la URL. Nosotros logramos dar un salto de calidad al incluir el *deep linking* en el **Capítulo 6**, identificando cada sección por medio de una URL única. Ahora bien, ¿qué sucede cuando, aun dentro de una sección, necesitamos acceder a una subsección o a un contenido específico?

Pensémoslo a modo de ejemplo: si visitamos el sitio **www.ejemplo.com/imagenes** y en la galería encontramos una imagen de nuestro interés, en caso de querer guardar el *link*, se nos presenta un problema: al acceder a la URL, volveremos a ver todas las imágenes cuando, en verdad, solo nos interesa una. Esta situación nos obligaría a tener que buscar entre todas las imágenes de la galería para poder visualizar la foto de nuestro interés. Lo que nosotros proponemos es una solución a este problema: identificar cada imagen con una URL independiente. Por ejemplo: **http://www.ejemplo.com/galeria/imagen1**.

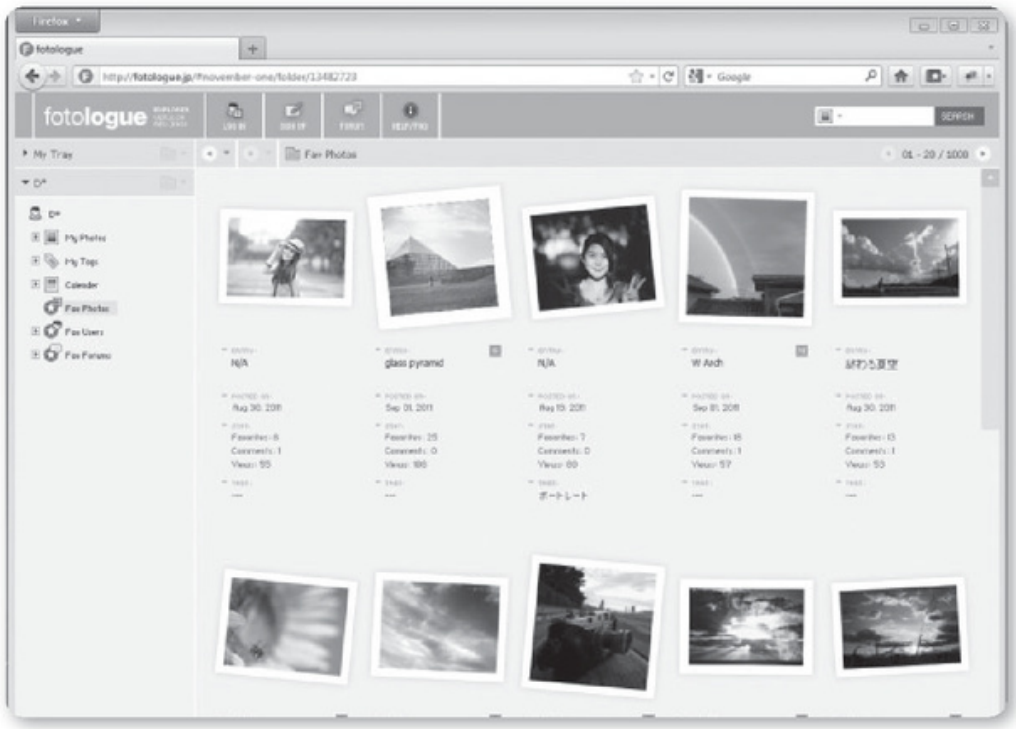



**Figura 19.** El sitio **`fotologue.jp`** es un ejemplo de navegación en Flash. Debemos visitarlo y ver la manera en que se trata el contenido.

O vayamos aún un paso más allá: ¿qué sucede si la imagen pertenece a una galería específica? Podríamos acceder a ella del siguiente modo: **http://www.ejemplo.com/galeria/seccion1/imagen1.**

A lo largo de este apartado, nos dedicaremos a realizar una galería de imágenes que contemple el caso de uso anteriormente mencionado, a fin de brindar a nuestros usuarios una experiencia más rica al momento de navegar el sitio.

> DEBEMOS OFRECER A NUESTROS VISITANTES UNA EXPERIENCIA MÁS RICA AL NAVEGAR

Teniendo en cuenta los principios de Programación Orientada a Objetos discutidos en el **Capítulo 5**, hemos decidido estructurar la galería de imágenes de la siguiente manera:

- **GalleryModel.as**: clase modelo de la galería que contendrá la información de cada imagen.
- **Gallery.as**: clase encargada de crear la grilla de imágenes.
- **AnimatedImage.as**: se trata de una clase gráfica útil para realizar la carga de las imágenes en miniatura.
- **DimmerImage.as**: clase que cargará las imágenes en tamaño completo.

# GalleryModel.as y gallery.xml

Lo primero que vamos a hacer para comenzar con el desarrollo de nuestra galería de imágenes es crear una estructura XML donde podamos almacenar la información de cada una de ellas. Cada nodo de imagen contiene lo mencionado a continuación:

- **path**: ruta de la imagen en miniatura.
- **name**: se trata de un pequeño texto que se hará visible al hacer **ROLL_OVER** sobre una miniatura de imagen.
- **largePath**: corresponde a la ruta de la imagen en tamaño completo que se descarga al hacer clic sobre una miniatura.

## OTROS USOS DEL DEEP LINKING

No perdamos de vista que, si bien en este caso trabajaremos con imágenes, el concepto es aplicable prácticamente a cualquier tema de desarrollo: desde galerías de video hasta reproductores de audio pueden aplicar esta lógica para la cual pensamos esta parte del *framework*.

Nuestro XML queda compuesto de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<gallery>
    <image>
        <path><![CDATA[imgs/1.jpg]]></path>
        <name><![CDATA[IMAGEN 1]]></name>
        <largePath><![CDATA[imgs/large/1.jpg]]></largePath>
    </image>
 </gallery>
```

En el **Capítulo 5** vimos los beneficios de usar el patrón MVC. Dentro de esta tríada, el modelo contiene la información de nuestra aplicación, de manera que se pueda acceder fácilmente desde cualquier clase.

Teniendo esto en cuenta, solo veremos aquellos métodos que sean específicos de **GalleryModel**:

```
public function getImageQuantity():Number{
    return _data.image.length();
}
public function getImagePath($index):String
{
    return _data.image[$index].path;
}
public function getName($index):String
{
    return _data.image[$index].name;
}
public function getLargePath($index):String {
    return _data.image[$index].largePath;

}
```

Solo necesitamos cuatro métodos (todos *getters*) para acceder a toda la información de la galería. El método **getImageQuantity()** nos permite acceder a la cantidad total de imágenes que se van a descargar.

Los tres métodos restantes nos devuelven información específica de la imagen sobre la base del índice que le indiquemos: **getImagePath()** nos devuelve la ruta de la imagen en miniatura; **getName()**, el nombre de la imagen; y **getLargePath()**, la ruta de la imagen en tamaño normal.

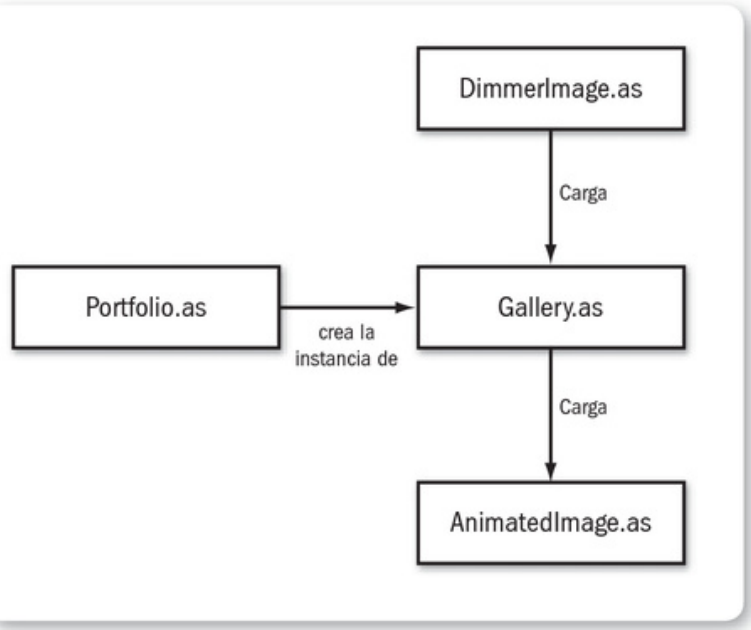



**Figura 20.** Diagrama de funcionamiento de la galería de imágenes para nuestro proyecto.

# PortfolioPage.as

La estructura de código que vamos a utilizar para la clase **PortfolioPage** es la misma que empleamos para **HomePage**: una vez hecho el **overwrite** del método **init()**, creamos la instancia de la clase **Gallery** y escuchamos por la carga de la estructura:

## ESTRUCTURA XML

Al pensar la estructura XML de nuestra galería, un buen recurso es analizar lo que hemos hecho a lo largo de este libro con el archivo **site.xml** de nuestro espacio de trabajo. La ventaja que nos brinda un archivo XML es que, con unos pocos pasos, podemos darle más funcionalidad a nuestro proyecto.

```
public class PortfolioPage extends AbstractSection

{
    private static var XML_PATH:String = "xml/gallery.xml";

    private var gallery:Gallery;

    public function PortfolioPage():void { }

    override public function init():void {

        GalleryModel.getInstance().addEventListener(Event.COMPLETE,
            onModelLoaded);
    GalleryModel.getInstance().load(XML_PATH);
}
```

## Gallery.as

Al completarse la carga del modelo de las imágenes (**onModelLoaded()**), creamos una instancia de la clase **Gallery** y la añadimos al escenario:

```
private function onModelLoaded(e:Event):void
{
    gallery = new Gallery();
    addChild(gallery);
}
```

### MODELO MVC Y DESARROLLO

Lo interesante de este ejemplo, independientemente del uso de la URL, es ver cómo podemos mantener la misma lógica de desarrollo del *framework* y aplicarla a otro recurso, como es una galería de imágenes. El patrón MVC nos sirve prácticamente para cualquier desarrollo visual.

La clase **Gallery** se encarga de agrupar en forma de grilla las miniaturas, teniendo en cuenta la cantidad total de imágenes que haya para mostrar.

Analicemos su constructor:

```
public function Gallery()
{
    addEventListener(Event.ADDED_TO_STAGE, onAdded);
    FrameworkSectionManager.getInstance().addEventListener(FrameworkSecti
        onManager.EXTRA_PARAMS_CHANGED, onExtraParams);
}
```

Lo primero que hacemos al instanciar esta clase es añadir dos *listeners*: el primero se encarga de avisarnos cuando la galería haya sido añadida al escenario, y el segundo nos permite ejecutar un evento cada vez que se modifiquen los parámetros adicionales de la URL. El segundo evento es el que nos interesa, porque es el que utilizaremos para realizar la carga y descarga de las imágenes en tamaño completo, y será ejecutado cada vez que un usuario haga clic sobre una miniatura.

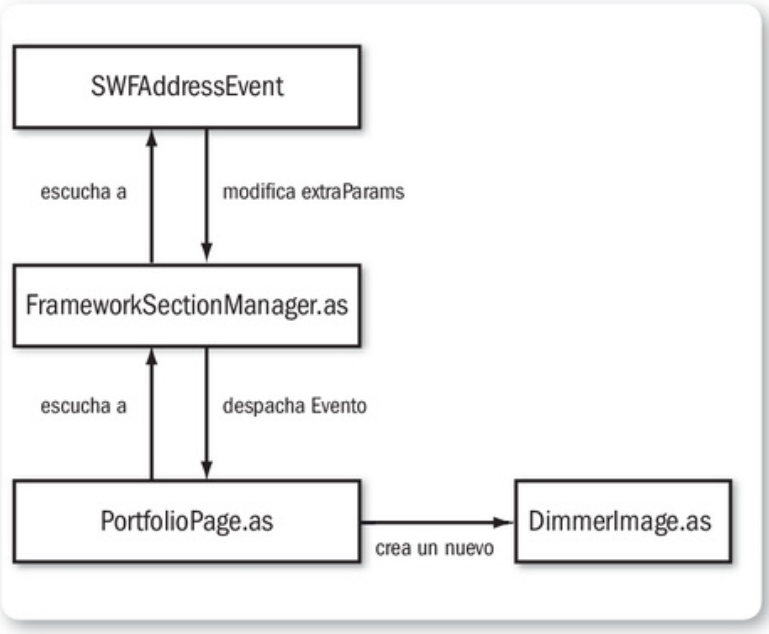



**Figura 21.** Diagrama de funcionamiento de **extraParams** para el correcto funcionamiento de la galería.

En la función **onAdded** vamos a crear un bucle que se ejecutará hasta que se alcance la cantidad de imágenes por descargar; por cada ciclo del bucle que se ejecute, almacenará en tres variables (**path, texto** y **largePath**) la información de cada uno de los nodos. Para esto haremos uso de las funciones que creamos antes en la clase **GalleryModel**. Por último, creamos una nueva instancia de **AnimatedImage** asignándole los parámetros necesarios y su correspondiente posición en el escenario:

```
private function onAdded(e:Event):void
{
    removeEventListener(Event.ADDED_TO_STAGE, onAdded);
    var currentRow:uint = 0;
    var currentColumn:uint = 0;
    for (var i:uint = 0; i < GalleryModel.getInstance().getImageQuantity(); i++) {
        var path:String = GalleryModel.getInstance().getImagePath(i);
        var texto:String = GalleryModel.getInstance().getName(i);
        var largeImagePath:String = GalleryModel.getInstance().getLargePath(i);
        var animImage:AnimatedImage = new AnimatedImage(path,
            texto,largeImagePath);
        addChild(animImage);
        TweenMax.from(animImage, 0.3, { alpha:0, rotation:10, delay: 0.1 * i });
        animImage.x = (290 + ESPACIADO_X) * currentColumn;
        animImage.y = (147 + ESPACIADO_Y) * currentRow;
        currentColumn ++;
        if (currentColumn > COLUMNAS - 1) {
            currentColumn = 0;
            currentRow++;
        }
    }
```

## PROGRAMACIÓN ORIENTADA A OBJETOS

Es importante recordar los conceptos que hemos visto a lo largo de este libro al trabajar con Programación Orientada a Objetos. Independientemente del funcionamiento de una clase, lo que nosotros necesitamos es poder acceder a su información, tanto para obtenerla como para modificarla.

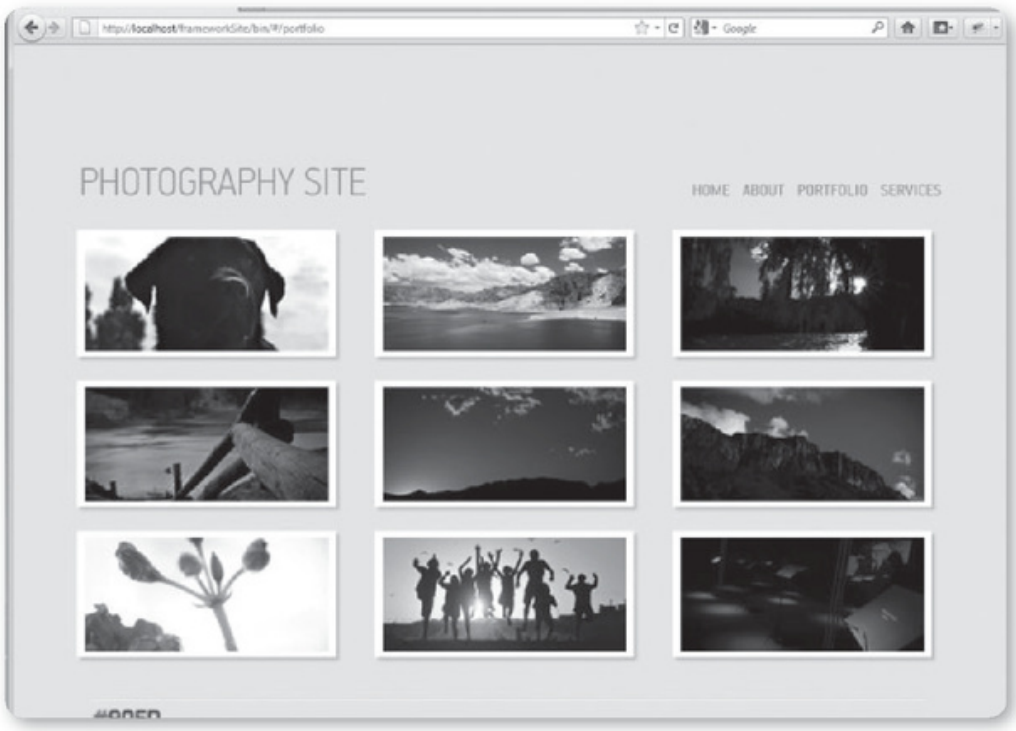



**Figura 22.** La galería de imágenes es creada por **Gallery**, que posiciona las instancias de **AnimatedImage** a modo de grilla.

## Averiguar si hay parámetros adicionales

Para saber si la URL a la cual se está accediendo tiene parámetros adicionales, debemos consultar el método **extraParams()** la clase **FrameworkSectionManager**. Si su largo es mayor que 1, sabemos que la URL tiene más de un parámetro:

```
var eParams:String = FrameworkSectionManager.getInstance().extraParams;
if(eParams.length > 1){
  onExtraParams();
}
```

En este caso, se va a ejecutar la función **onExtraParams()**, dentro de la cual parseamos la información y se la pasamos a la instancia de la clase **DimmerImage** la cual se encarga de crear un rectángulo negro del tamaño de la pantalla, asignarle una transparencia y posicionar la imagen en el centro del mismo.

```
private function onExtraParams(e:Event = null):void
{
    var imagePath:String = FrameworkSectionManager.getInstance().
        extraParams;
    var arrayUrl:Array = imagePath.split("/");
    var url:String = arrayUrl[2] + "/" + arrayUrl[1] + "/" + arrayUrl[0];
    var dimmerImage:DimmerImage = new DimmerImage(url);
    stage.addChild(dimmerImage);
}
```

# AnimatedImage.as

En la clase **Gallery** creamos una instancia de **AnimatedImage** por cada nodo imagen que se había almacenado en el archivo XML.

Como veremos a continuación, lo primero que debemos hacer en esta clase es almacenar en variables locales la información proporcionada anteriormente, para poder hacer la carga de la imagen requerida y definir el texto que se hará visible en el **ROLL_OVER**. Una vez almacenadas las variables, añadimos un evento que se ejecutará cuando el elemento sea añadido al escenario:

```
public class AnimatedImage extends MovieClip
    {
    // path de la image en minatura
    private var _path:String;
    // texto a mostrar en el rollOver
    private var _text:String;
    // clip utilizado como backgorund del boton
    private var background:ImgBackground;
    // contenedor de la imagen
    private var imgContainer:MovieClip;
    // mascara de la imagen
    private var imgMask:Sprite;
    // loader de la imagen

    }
```

```
        private var imgLoader:Loader;
        // urlRequest para el path de la imagen
        private var urlRequest:URLRequest
        // Clip que contiene el texto de la imagen
        private var imgText:ImgText;
        // PAth de la imagen en tamaño completo.
        private var _largeImgPath:String;

    public function AnimatedImage(path:String, text:String, largeImgPath:String)
        {
            _path = path;
            _text = text;
            _largeImgPath = largeImgPath;
            addEventListener(Event.ADDED_TO_STAGE, onAdded);
```

Cuando **AnimatedImage** pasa a formar parte del escenario, le creamos

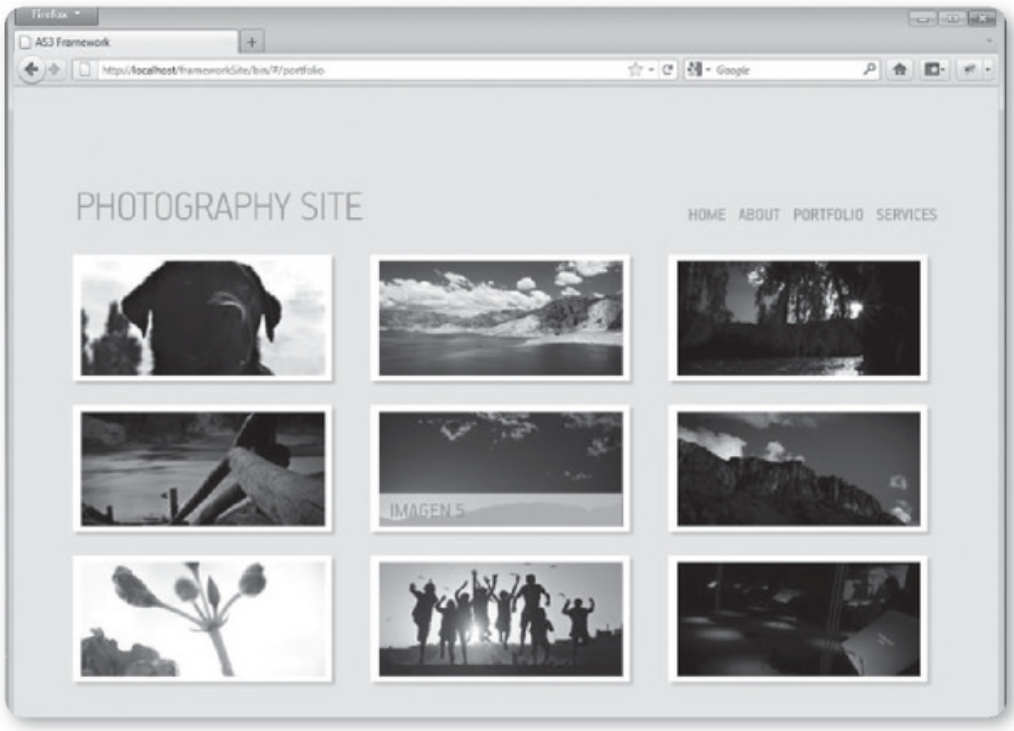



**Figura 23.** Comportamiento de **ROLL_OVER** del **AnimatedImage**: al posicionarnos sobre ella, mostramos su nombre.

su respectivo *loader* (clase **Loader**) y comenzamos la carga de la imagen. Es necesario crear un *listener* que se ejecute cuando se finalice la carga: cuando sabemos que la imagen se cargó, podemos asignarle el resto de las funcionalidades:

```
private function onAdded(e:Event):void
{
    removeEventListener(Event.ADDED_TO_STAGE, onAdded);
    this.buttonMode = true;
    imgLoader = new Loader();
    urlRequest = new URLRequest(_path);
    imgLoader.contentLoaderInfo.addEventListener(Event.COMPLETE,
        onCompletLoading);
    imgLoader.load(urlRequest);
}
```

Una vez cargada la imagen, podemos crear el resto de los elementos del **AnimatedImage**. Para hacerlo, vamos a instanciar el clip **ImgBackground**, que se encuentra almacenado en la biblioteca y utilizaremos como fondo de nuestra imagen. Luego, instanciaremos el clip **ImgText**, que posee el campo de texto donde mostraremos el nombre de la imagen cuando se haga **ROLL_OVER**:

```
private function onCompletLoading(e:Event):void
{
    background = new ImgBackground();
    addChild(background);
    imgContainer = new MovieClip();
    imgContainer.addChild(imgLoader);
    imgContainer.x = 9;
    imgContainer.y = 7;
    addChild(imgContainer);

    imgText = new ImgText();
    imgText.nameTxt.text = _text;
    imgText.x = 7;
```

```
        imgText.y = imgContainer.y + imgContainer.height;
        addChild(imgText);
    }
```

Por último, creamos una máscara que permita ocultar el contenido del campo de texto cuando se ejecute un evento **ROLL_OVER**. Para hacerlo, vamos a hacer uso de la API de dibujo de Flash para crear un rectángulo y asignárselo como máscara al clip **imgText**:

```
imgMask = new Sprite();
imgMask.graphics.lineStyle(1, 0xFFFFFF);
imgMask.graphics.beginFill(0xFFFFFF);
imgMask.graphics.drawRect(0, 0, 270, 127);
imgMask.graphics.endFill();
imgMask.x = 9;
imgMask.y = 8;
addChild(imgMask);
imgText.mask = imgMask;

this.addEventListener(MouseEvent.MOUSE_OVER, onOver);
this.addEventListener(MouseEvent.CLICK, onClick);
this.addEventListener(MouseEvent.MOUSE_OUT, onOut);
```

Para finalizar con esta clase, solo nos resta crear la función que se ejecute en el evento **CLICK**. Esta utilizará el método **setValue()** de la clase **FrameworkSiteManager**, pasando como parámetros la sección actual y la URL de la imagen completa a modo de parámetro especial. Cuando esto suceda, se ejecutará la función **onExtraParams()** de la clase **Gallery**, que se ocupará de cargar la imagen en tamaño completo.

# DimmerImage.as

La clase **DimmerImage** crea un recuadro negro del tamaño del escenario y carga una imagen en tamaño completo sobre la base del parámetro que se le haya pasado:

```
var dimmerImage:DimmerImage = new DimmerImage(url);
stage.addChild(dimmerImage);
```


**Figura 24.** La imagen se encuentra dentro de **`DimmerImage`** y la vemos cada vez que cliqueamos un elemento **`AnimatedImage`**.

La instancia de **DimmerImage** se añade directamente sobre el escenario, ya que necesitamos que la imagen se cargue por encima del resto de nuestro sitio, con el fin de brindarle mayor protagonismo y eliminar la interacción con el resto de la galería.

## DIBUJO POR CÓDIGO Y DESDE LA IDE

En el capítulo sobre performance, indicamos la diferencia de estos dos métodos de dibujo y explicamos cuándo resulta ventajoso emplear uno u otro. Estos dos métodos pueden convivir tranquilamente: en nuestra galería de imágenes, estamos haciendo uso de ambas metodologías.

# ¡Llegamos al fin!

Un capítulo de este libro basta para poner a prueba todo lo que hemos explicado a lo largo de estas páginas. Para aquellos programadores avanzados seguramente sea una experiencia gratificante ver cómo, con unas pocas líneas, podemos implementar soluciones reutilizables. A aquellos lectores a quienes les resulte complejo comprender la lógica de este proceso, les recomendamos no desanimarse: si bien existe un importante nivel de complejidad en este desarrollo, es necesario afrontarlo con tiempo y paciencia. Cuando las cosas no salgan, habrá que volver a las fuentes, analizar con tranquilidad, y pasar tiempo en compañía de estas páginas y de los archivos que utilizamos: independientemente de nuestras explicaciones, ver el código y su funcionamiento muchas veces es una solución mucho más clarificadora.

## RESUMEN

A lo largo de este capítulo vimos lo sencillo que resulta realizar un proyecto teniendo un espacio de trabajo propio al cual relegar los procesos de desarrollo reiterativos y comunes a cada uno. Esta metodología nos permite abocarnos a lo específico de cada trabajo, a sabiendas de que las funcionalidades principales se realizan de manera automática por un entorno que fue creado y optimizado para ese fin.

# Actividades

## TEST DE AUTOEVALUACIÓN

| | |
|---|---|
| 1 | ¿Cuál es la clase principal que debemos extender al comenzar un proyecto? |
| 2 | ¿Para qué nos sirve añadir un listener que ejecute el evento **Event.ADDED_TO_STAGE**? |
| 3 | ¿Cuál es la ventaja de utilizar *deep linking* en una galería de imágenes? |
| 4 | ¿A qué clase debemos extender si deseamos crear una nueva sección? |
| 5 | ¿Cuáles son las ventajas de utilizar una clase Modelo para la creación de una galería de imágenes? ¿Qué similitudes hay con el modelo del framework? |
| 6 | ¿Qué es el *Liquid Layout*? ¿Cuáles son las ventajas de utilizarlo? ¿Qué eventos debemos escuchar para poder crear uno? |
| 7 | ¿Cuál es la ventaja de sobrescribir funciones? ¿Para qué sirve el método **super()**? |
| 8 | ¿Qué función debemos utilizar para cambiar de sección? |
| 9 | ¿De qué manera podemos enviar parámetros especiales? ¿Cómo podemos notificar a nuestra aplicación cuando se ejecutó un evento de parámetros especiales? |
| 10 | ¿Qué clases debemos extender para la creación de un preloader? |

## ACTIVIDADES PRÁCTICAS

| | |
|---|---|
| 1 | Añada una nueva sección en el **site.xml** e impleméntela en el sitio. |
| 2 | Cree un *Liquid Layout* con comportamientos específicos para cada sección. |
| 3 | Cree dos o más subsecciones en la sección de **Servicios**. |
| 4 | Añada un sistema de paginado en la clase **Gallery.as**. |
| 5 | Utilizando los datos del XML, cree una versión alternativa a la galería para los casos en que el usuario no posea FlashPlayer. |

# Servicios al lector

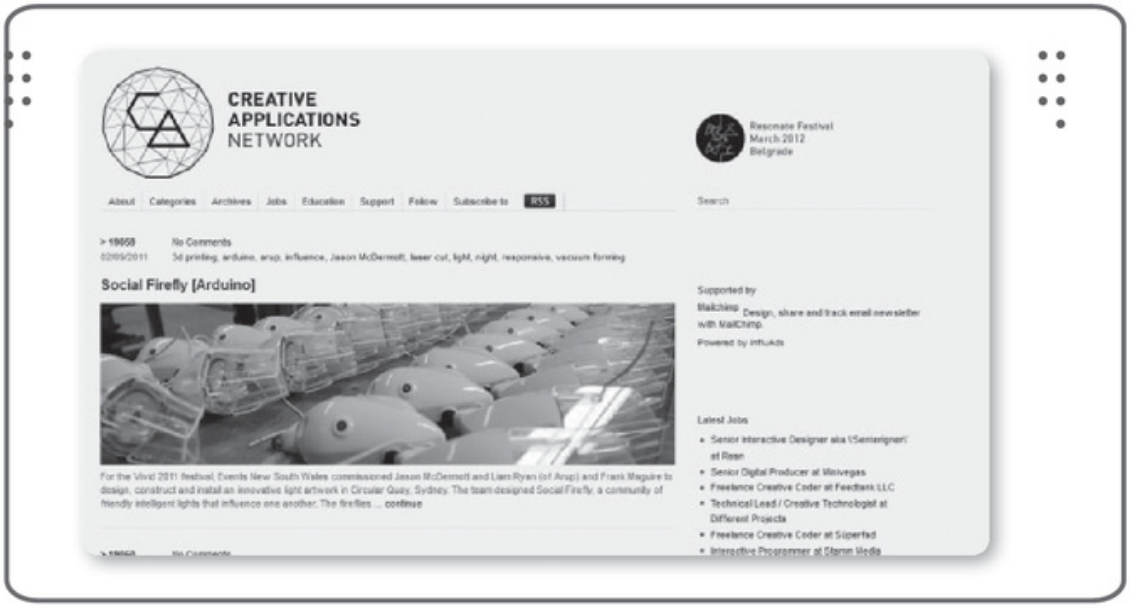
En esta sección nos encargaremos de presentar un útil índice temático para que podamos encontrar en forma sencilla los términos que necesitamos. Además, podremos ver una interesante selección de sitios y programas que están relacionados con el contenido de esta obra.

# Índice temático

# Sitios web relacionados

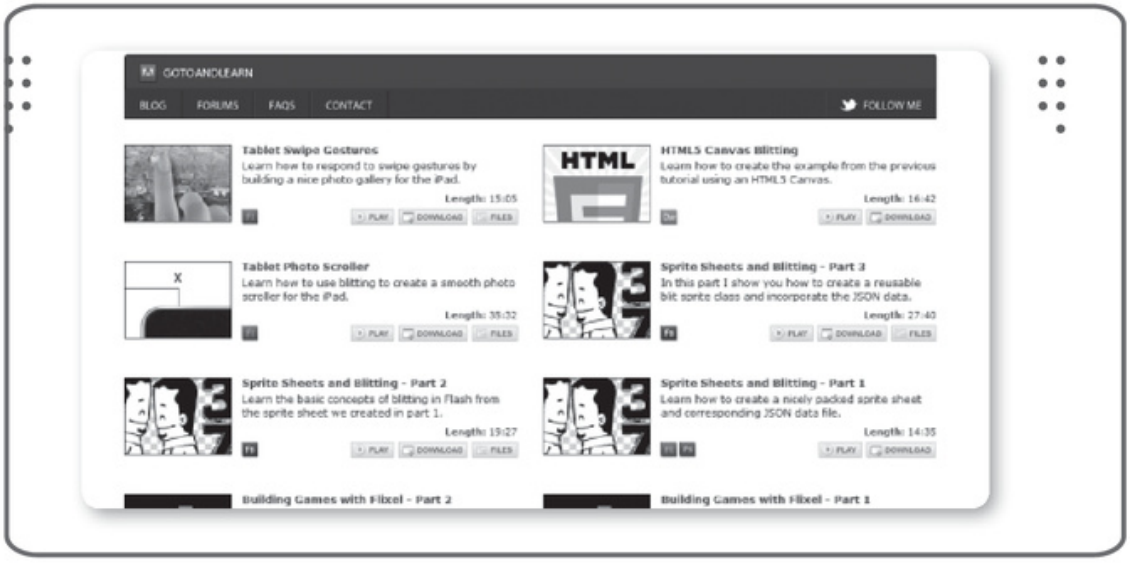
## CREATIVE APPLICATIONS NETWORK ● www.creativeapplications.net

Sitio web que presenta la más variada cantidad de desarrollos, la mayoría de ellos, de carácter experimental. Si bien hay muchos hechos en Flash, es un muy buen lugar para encontrar desarrollos que impliquen tecnologías como Processing, OpenFrameworks y Arduino.




## GOTOANDLEARN ● www.gotoandlearn.com

Sitio de Lee Brimelow con videotutoriales sobre el mundo Flash, Flex y AIR. Es posible descargar los archivos que se utilizan de ejemplo, y si bien los videos están en inglés, se explican de una manera muy clara y legible. Twitter: http://twitter.com/#!/leebrimelow.

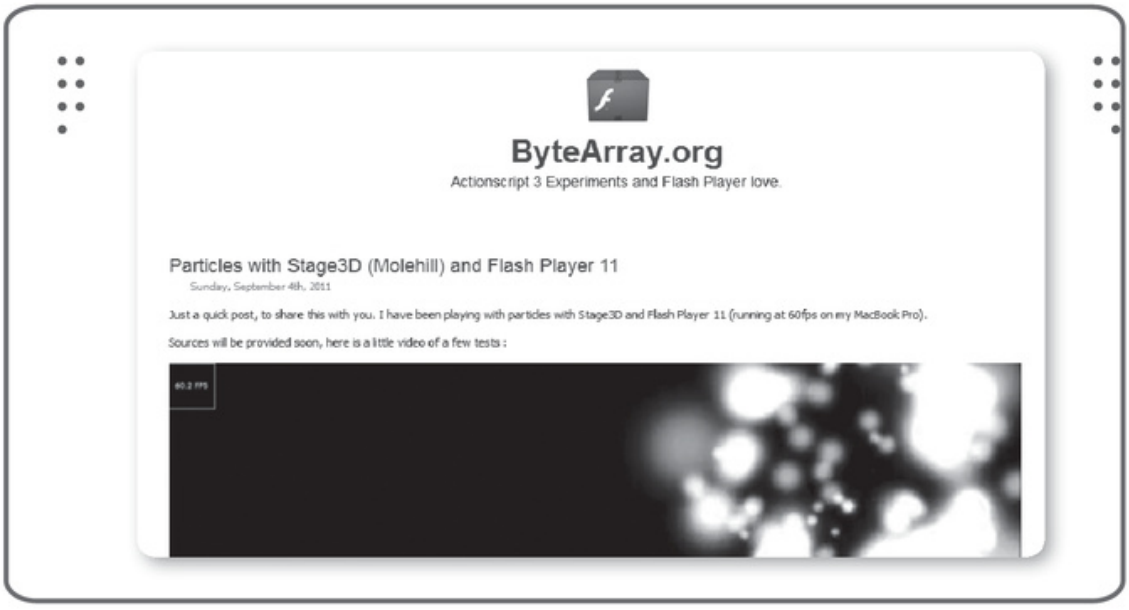


## ANDRE MICHELLE ● www.andre-michelle.com

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 www.andre-michelle.com. Twitter: http://twitter.com/#!/andremichelle.

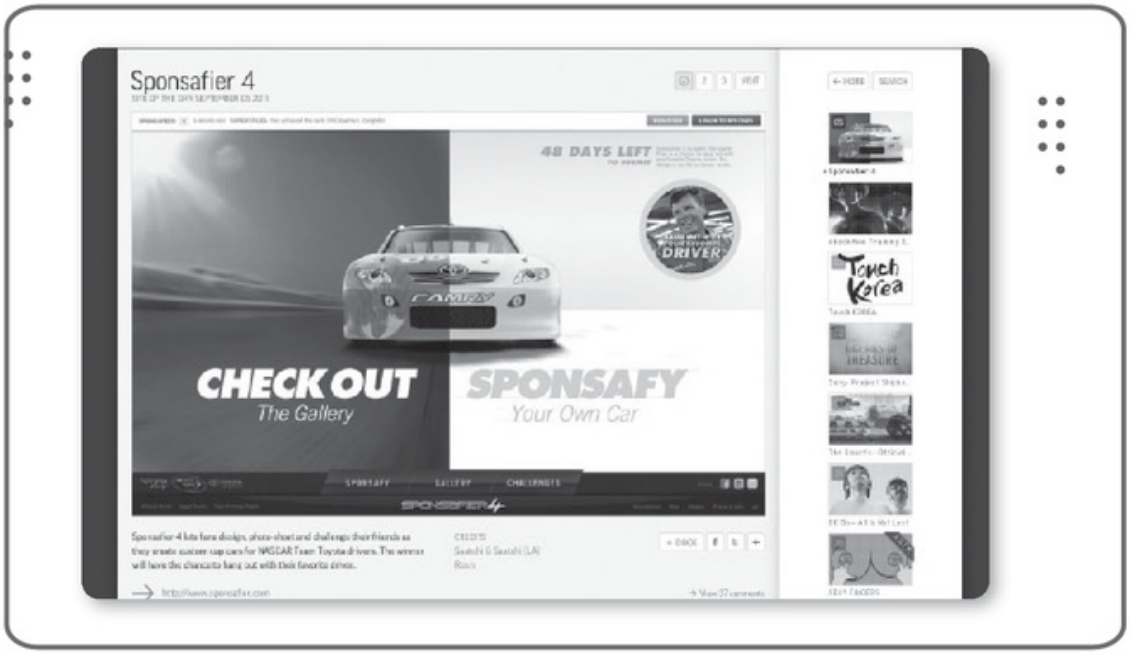

## BYTE ARRAY ● www.bytearray.org

Sitio web de Thibault Imbert. Si bien contiene mucho material de carácter experimental, es una muy buena fuente de información sobre la actualidad de Flash y la salida de nuevos recursos para la plataforma. Twitter: http://twitter.com/#!/thibault_Imbert.



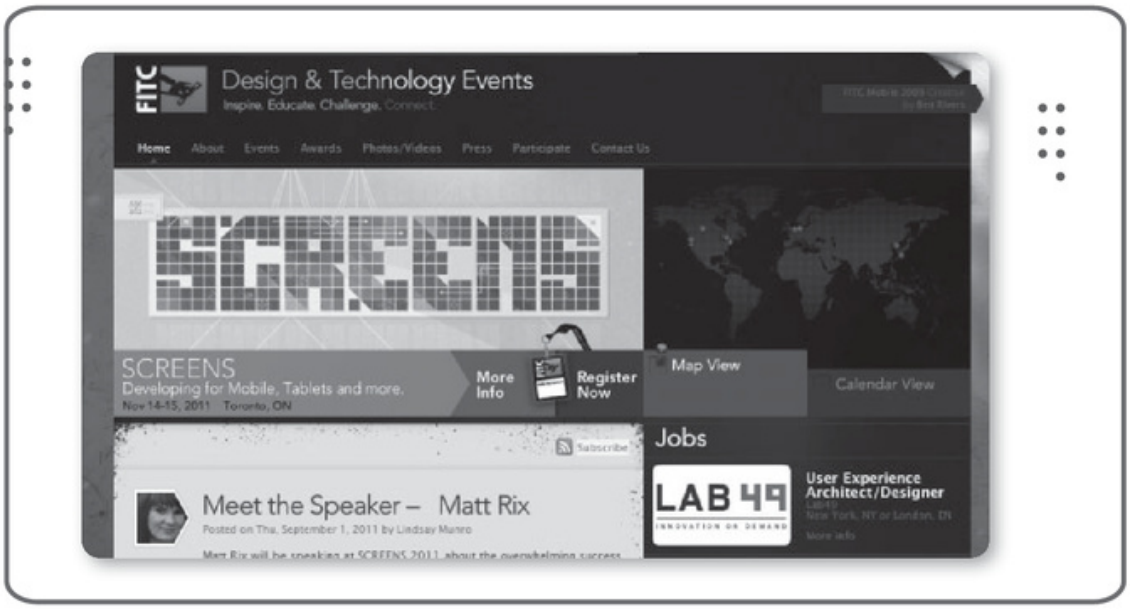
## THE FWA ● www.thefwa.com

The FWA (*The 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 se galardona a la mejor web del mes y a la mejor del año. Es un buen lugar para encontrar tanto sitios como aplicaciones vanguardistas.



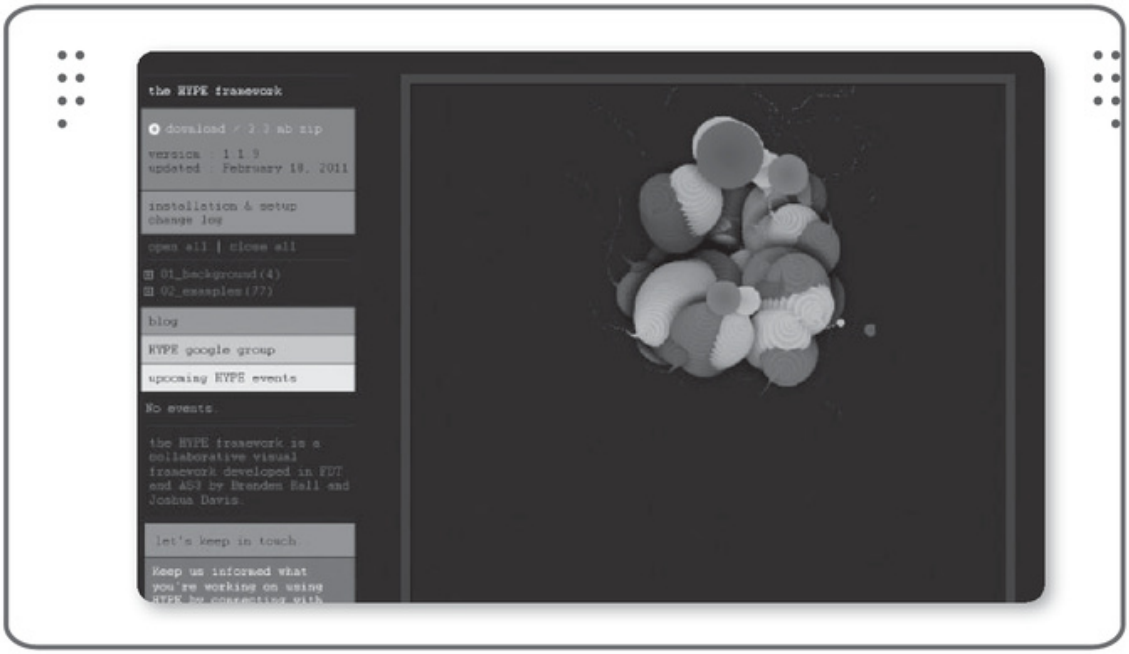

## FITC ● www.fitc.ca

FITC organiza eventos sobre diseño y tecnología con el objetivo de educar, proponer nuevas metas y conectar a los desarrolladores. Además de su sitio web, recomendamos seguirlo a través de su cuenta en Twitter o de su grupo en Facebook. Twitter: http://twitter.com/#!/FITC.



## HYPE FRAMEWORK ● www.hypeframework.org

Hype es un framework creado en ActionScript 3.0 por Branden Hall y Joshua Davis, dos reconocidos desarrolladores de Flash. Es importante tener en cuenta que se trata de una opción orientada, principalmente, a la creación de efectos visuales y sonoros.




## SOULWIRE ● http://blog.soulwire.co.uk

Soulwire es un blog de arte y tecnología creado por Justin Windle, artista, diseñador y creativo. Su sitio se destaca por el inmenso potencial con el que cuenta para desarrollos experimentales. La mayoría de los códigos que emplea están disponibles. Muy recomendable.

## 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 tapa, pregunte y compare.

### 4 ¿TIENE VALORES AGREGADOS?

Desde un sitio exclusivo en la Red, un Servicio de Atención al Lector, la posibilidad de leer el sumario en la Web para evaluar con tranquilidad la compra, y hasta la presencia de adecuados índices temáticos, todo suma al valor de un buen libro.

### 5 VERIFIQUE EL IDIOMA

No solo el del texto; también revise que las pantallas incluidas en el libro estén en el mismo idioma del programa que usted utiliza.

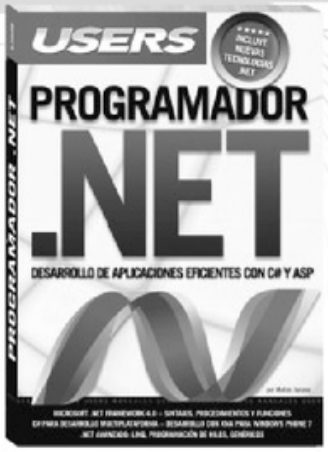
# ¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.


### Microcontroladores

Este manual es ideal para todos aquellos que quieran iniciarse en la programación de microcontroladores. A través de esta obra, podrán aprender sobre los microcontroladores PIC 16F y 18F, hasta llegar a conectar los dispositivos de forma inalámbrica, entre muchos otros proyectos.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / 978-987-1773-56-5




### Programador .NET

Este libro está dirigido a todos aquellos que quieran iniciarse en el desarrollo bajo lenguajes Microsoft. A través de los capítulos del manual, aprenderemos sobre POO, la programación con tecnologías .NET y de qué manera se desenvuelven con otras tecnologías existentes.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / 978-987-1773-26-8


### Photoshop: proyectos y secretos

En esta obra aprenderemos a utilizar Photoshop, desde la original mirada de la autora. Con el foco puesto en la comunicación visual, a lo largo del libro adquiriremos conocimientos teóricos y prácticos, para conocer los efectos y herramientas que ofrece el programa.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / 978-987-1773-25-1


### WordPress

Este manual está dirigido a todos aquellos que quieran presentar sus contenidos o los de sus clientes a través de WordPress. En sus páginas el autor nos enseñará desde cómo llevar adelante la administración del blog hasta las posibilidades de interacción con las redes sociales.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / 978-987-1773-18-3


### Administrador de servidores

Este libro es la puerta de acceso para ingresar en el apasionante mundo de los servidores. Aprenderemos desde los primeros pasos sobre la instalación, configuración, seguridad y virtualización, hasta tener el control de los servidores en la palma de nuestras manos.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-19-0


### Windows 7: Trucos y secretos

Este libro está dirigido a todos aquellos que quieran sacar el máximo provecho de Windows 7, las redes sociales y los dispositivos ultraportátiles del momento. A lo largo de sus páginas, el lector podrá adentrarse en estas tecnologías mediante trucos inéditos y consejos asombrosos.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-17-6


### Desarrollo PHP + MySQL

Este libro presenta la fusión de dos de las herramientas más populares para el desarrollo de aplicaciones web de la actualidad: PHP y MySQL. En sus páginas, el autor nos enseñará las funciones del lenguaje para aplicar lo aprendido en nuestros propios desarrollos.

→ COLECCIÓN: MANUALES USERS
→ 432 páginas / ISBN 978-987-1773-16-9


### Excel 2010

Este manual resulta ideal tanto para quienes se inician en el uso de Excel, como para los usuarios que quieran conocer las nuevas herramientas que ofrece la versión 2010. La autora nos enseñará desde cómo ingresar datos hasta la forma de imprimir ahorrando papel y tiempo.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-15-2


### Técnico Hardware

Esta obra es una completa guía para aprender a llevar adelante un correcto diagnóstico y determinar la solución más adecuada para los problemas de hardware. En sus páginas veremos todas las técnicas necesarias para implementar las soluciones de los profesionales.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-14-5


### PHP Avanzado

Este libro brinda todas las herramientas necesarias para acercar al trabajo diario del desarrollador los avances más importantes incorporados en PHP 6. En sus páginas, repasaremos todas las técnicas actuales para potenciar el desarrollo de sitios web.

→ COLECCIÓN: MANUALES USERS
→ 400 páginas / ISBN 978-987-1773-07-7


### AutoCAD

Este manual nos presenta un recorrido exhaustivo por el programa más difundido en dibujo asistido por computadora a nivel mundial, en su versión 2010. En sus páginas, aprenderemos desde cómo trabajar con dibujos predeterminados hasta la realización de objetos 3D.

→ COLECCIÓN: MANUALES USERS
→ 384 páginas / ISBN 978-987-1773-06-0


### Windows 7 Avanzado

Esta obra nos presenta un recorrido exhaustivo que nos permitirá acceder a un nuevo nivel de complejidad en el uso de Windows 7. Todas las herramientas son desarrolladas con el objetivo de acompañar al lector en el camino a ser un usuario experto.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-08-4

USERS

## CONTENIDO



NIVEL DE USUARIO

PRINCIPIANTE | INTERMEDIO | AVANZADO | EXPERTO

# FLASH

## DESARROLLO PROFESIONAL

Esta obra se encuentra destinada a todos los desarrolladores que necesitan avanzar en el uso de la Plataforma Adobe Flash para sacar el máximo provecho en la ejecución de sus proyectos.

A través de los distintos capítulos, los autores presentan los temas más importantes para tener en cuenta al momento de llevar adelante proyectos profesionales en ActionScript 3.0. Para esto, se explica cómo crear un framework de trabajo personalizado a través de diferentes técnicas y conceptos: OOP, MVC, Search Engine Optimization, usabilidad, performance, desarrollo para dispositivos, y mucho más.

Al terminar el libro, el lector podrá implementar un caso real, en el que se integra y se pone en práctica todo lo visto; una forma ideal de concluir la obra, en donde el desarrollador alcanzará el grado de experto en ActionScript 3.0 y Flash CS5.

En RedUSERS.com 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.

Si desea más información sobre el libro puede comunicarse con nuestro Servicio de Atención al Lector: usershop@redusers.com

### FLASH PROFESSIONAL DEVELOPMENT

This manual offers innovative techniques and concepts for Flash developers, who need to get an in-depth sight on the Adobe platform. Learn and master ActionScript 3.0 like professionals do, with this unique book.


ISBN 978-987-1857-00-5


MANUALES USERS MANUALES USERS MANUALES

RedUSERS.com

TÉCNICAS AVANZADAS Y BUENAS PRÁCTICAS