Programación con ADOBE® ACTIONSCRIPT® 3.0

Adobe

Programación con Adobe® ActionScript® 3.0 para Adobe® Flash®

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA

Contenido

Capítulo 1: Información sobre este manual

En este manual se ofrecen unas bases para el desarrollo de aplicaciones en Adobe® ActionScript® 3.0. Para comprender mejor las ideas y las técnicas descritas, debe estar familiarizado con conceptos de programación generales como, por ejemplo, tipos de datos, variables, bucles y funciones. También hay que conocer conceptos básicos sobre la programación orientada a objetos, como los conceptos de clase y herencia. Un conocimiento previo de ActionScript 1.0 o ActionScript 2.0 es útil pero no necesario.

Utilización de este manual

Los capítulos de este manual están organizados en los siguientes grupos lógicos para ayudarle a encontrar áreas relacionadas de la documentación de ActionScript:

Capítulo	Descripción
En los capítulos 2 a 5 se ofrece información general sobre la programación en ActionScript.	Se abordan los conceptos básicos de ActionScript 3.0, incluida la sintaxis del lenguaje, sentencias, operadores y programación ActionScript orientada a objetos.
En los capítulos 6 a 11, se describen las clases y los tipos de datos básicos de ActionScript 3.0.	Se describen los tipos de datos de nivel superior de ActionScript 3.0.
En los capítulos 12 a 32, se describen las API de Flash Player y de Adobe AIR.	Se describen funciones importantes implementadas en paquetes y clases específicos de Adobe Flash Player y Adobe AIR, como el control de eventos, los objetos y la lista de visualización, las conexiones de red y las comunicaciones, la entrada y salida de archivos, la interfaz externa, el modelo de seguridad de aplicaciones, etc.

Este manual también contiene numerosos archivos de ejemplo que ilustran conceptos de programación de aplicaciones para clases importantes o utilizadas frecuentemente. Los archivos de ejemplo se empaquetan de forma que resulten fáciles de cargar y utilizar con Adobe® Flash® CS4 Professional y pueden incluir archivos envolventes. Sin embargo, el núcleo del código de ejemplo es puro código ActionScript 3.0 que se puede utilizar en el entorno de desarrollo que se prefiera.

Se puede escribir y compilar código ActionScript 3.0 de varias maneras:

- Mediante el entorno de desarrollo Adobe Flex Builder 3.
- Utilizando cualquier editor de texto y un compilador de línea de comandos, como el proporcionado con Flex Builder 3.
- Usando la herramienta de edición Adobe ® Flash® CS4 Professional.

Para obtener más información sobre los entornos de desarrollo de ActionScript, consulte "Introducción a ActionScript 3.0" en la página 4

Para comprender los ejemplos de código de este manual no es necesario tener experiencia previa en el uso de entornos de desarrollo integrados para ActionScript, como Flex Builder o la herramienta de edición de Flash. No obstante, es conveniente consultar la documentación de estas herramientas para aprender a utilizarlas y a escribir y compilar código ActionScript 3.0. Para obtener más información, consulte "Acceso a la documentación de ActionScript" en la página 2.

Acceso a la documentación de ActionScript

Este manual se centra en describir ActionScript 3.0, un completo y eficaz lenguaje de programación orientado a objetos. No incluye una descripción detallada del proceso de desarrollo de aplicaciones ni del flujo de trabajo en una herramienta o arquitectura de servidor específica. Por ello, además de leer *Programación con ActionScript 3.0*, es recomendable consultar otras fuentes de documentación al diseñar, desarrollar, probar e implementar aplicaciones de ActionScript 3.0.

Documentación de ActionScript 3.0

En este manual se explican los conceptos relacionados con la programación en ActionScript 3.0, se muestran los detalles de implementación y se proporcionan ejemplos que ilustran las características importantes del lenguaje. No obstante, este manual no es una referencia del lenguaje exhaustiva. Para una referencia completa, consulte la Referencia del lenguaje y componentes ActionScript 3.0, en la que se describen todas las clases, métodos, propiedades y eventos del lenguaje. La Referencia del lenguaje y componentes ActionScript 3.0 proporciona información de referencia detallada sobre el núcleo del lenguaje, los componentes de la herramienta de edición de Flash (paquetes fl) y las API de Flash Player y de Adobe AIR (paquetes flash).

Documentación de Flash

Si utiliza la herramienta de edición de Flash, es posible que desee consultar estos manuales:

Manual	**Descripción**
Utilización de Flash	Describe la manera de desarrollar aplicaciones Web dinámicas en la herramienta de edición de Flash.
Programación con ActionScript 3.0	Describe el uso específico del lenguaje ActionScript 3.0 y la API principal de Flash Player y de Adobe AIR
Referencia del lenguaje y componentes ActionScript 3.0	Proporciona información sobre la sintaxis y el uso, así como ejemplos de código, para los componentes de la herramienta de edición de Flash y la API de ActionScript 3.0.
Utilización de componentes ActionScript 3.0	Explica los detalles del uso de componentes para desarrollar aplicaciones creadas por Flash.
Desarrollo de aplicaciones de Adobe AIR con Flash CS4 Professional	Describe cómo desarrollar e implementar aplicaciones de Adobe AIR utilizando ActionScript 3.0 y la API de Adobe AIR en Flash
Aprendizaje de ActionScript 2.0 en Adobe Flash	Proporciona información general sobre la sintaxis de ActionScript 2.0 y sobre cómo utilizar ActionScript 2.0 al trabajar con distintos tipos de objetos
Referencia del lenguaje ActionScript 2.0	Proporciona información sobre la sintaxis y el uso, así como ejemplos de código, para los componentes de la herramienta de edición de Flash y la API de ActionScript 2.0.
Utilización de componentes ActionScript 2.0	Explica de forma detallada cómo utilizar componentes de ActionScript 2.0 para desarrollar aplicaciones creadas por Flash.
Referencia del lenguaje de componentes ActionScript 2.0	Describe cada componente disponible en la versión 2 de la arquitectura de componentes de Adobe, junto con su API
Ampliación de Flash	Describe los objetos, métodos y propiedades disponibles en la API de JavaScript
Introducción a Flash Lite 2.x	Explica cómo utilizar Adobe® Flash® Lite™ 2.x para desarrollar aplicaciones y proporciona información sobre la sintaxis y el uso, así como ejemplos de código para las funciones de ActionScript disponibles en Flash Lite 2.x

Manual	Descripción
Desarrollo de aplicaciones de Flash Lite 2.x	Explica cómo desarrollar aplicaciones de Flash Lite 2.x
Introducción a ActionScript en Flash Lite 2.x	Ofrece una introducción al desarrollo de aplicaciones Flash Lite 2.x y describe todas las funciones de ActionScript disponibles para los desarrolladores de Flash Lite 2.x
Referencia del lenguaje ActionScript de Flash Lite 2.x	Proporciona información sobre la sintaxis y el uso, así como ejemplos de código, de la API de ActionScript 2.0 disponible en Flash Lite 2.x
Introducción a Flash Lite 1.x	Proporciona una introducción a Flash Lite 1.x y describe el modo de comprobar contenido con el emulador de Adobe® Device Central CS4
Desarrollo de aplicaciones de Flash Lite 1.x	Describe la manera de desarrollar aplicaciones para dispositivos móviles con Flash Lite 1.x
Aprendizaje de ActionScript en Flash Lite 1.x	Explica cómo utilizar ActionScript en aplicaciones de Flash Lite 1.x y describe todas las funciones de ActionScript disponibles en Flash Lite 1.x
Referencia del lenguaje ActionScript de Flash Lite 1.x	Describe la sintaxis y el modo de uso de los elementos de ActionScript disponibles en Flash Lite 1.x

Recursos de aprendizaje de ActionScript

Además del contenido de estos manuales, Adobe proporciona regularmente artículos actualizados, ideas de diseño y ejemplos en el Centro de desarrollo de Adobe y el Centro de diseño de Adobe.

Centro de desarrollo de Adobe

El Centro de desarrollo de Adobe contiene la información más actualizada sobre ActionScript, artículos sobre el desarrollo de aplicaciones reales e información sobre nuevos problemas importantes. Consulte el Centro de desarrollo de Adobe en http://www.adobe.com/es/devnet/.

Centro de diseño de Adobe

Póngase al día en diseño digital y gráficos en movimiento. Examine la obra de importantes artistas, descubra las nuevas tendencias de diseño y mejore sus conocimientos con tutoriales, flujos de trabajo clave y técnicas avanzadas. Consulte el centro cada quince días para ver tutoriales y artículos nuevos e inspirarse con las creaciones de las galerías. Consulte el centro de diseño en www.adobe.com/designcenter/.

Capítulo 2: Introducción a ActionScript 3.0

En este capítulo se ofrece información general sobre Adobe® ActionScript® 3.0, una de las versiones más recientes e innovadoras de ActionScript.

ActionScript

ActionScript es el lenguaje de programación para los entornos de tiempo de ejecución de Adobe® Flash® Player y Adobe® AIR™. Entre otras muchas cosas, activa la interactividad y la gestión de datos en el contenido y las aplicaciones de Flash, Flex y AIR.

ActionScript se ejecuta mediante la máquina virtual ActionScript (AVM), que forma parte de Flash Player y AIR. El código de ActionScript se suele compilar en un *formato de código de bytes* (un tipo de lenguaje que los ordenadores pueden escribir y comprender) mediante un compilador, como el incorporado en Adobe® Flash® CS4 Professional o Adobe® Flex™ Builder™ o el que está disponible en el SDK de Adobe® Flex™. El código de bytes está incorporado en los archivos SWF ejecutados por Flash Player y AIR.

ActionScript 3.0 ofrece un modelo de programación robusto que resultará familiar a los desarrolladores con conocimientos básicos sobre programación orientada a objetos. Algunas de las principales funciones de ActionScript 3.0 que mejoran las versiones anteriores son:

- Una nueva máquina virtual de ActionScript, denominada AVM2, que utiliza un nuevo conjunto de instrucciones de código de bytes y proporciona importantes mejoras de rendimiento.
- Una base de código de compilador más moderna que realiza mejores optimizaciones que las versiones anteriores del compilador.
- Una interfaz de programación de aplicaciones (API) ampliada y mejorada, con un control de bajo nivel de los objetos y un auténtico modelo orientado a objetos.
- Una API XML basada en la especificación de ECMAScript para XML (E4X) (ECMA-357 edición 2). E4X es una extensión del lenguaje ECMAScript que añade XML como un tipo de datos nativo del lenguaje.
- Un modelo de eventos basado en la especificación de eventos DOM (modelo de objetos de documento) de nivel 3.

Ventajas de ActionScript 3.0

ActionScript 3.0 aumenta las posibilidades de creación de scripts de las versiones anteriores de ActionScript. Se ha diseñado para facilitar la creación de aplicaciones muy complejas con conjuntos de datos voluminosos y bases de código reutilizables y orientadas a objetos. Aunque no se requiere para el contenido que se ejecuta en Adobe Flash Player, ActionScript 3.0 permite introducir unas mejoras de rendimiento que sólo están disponibles con AVM2, la nueva máquina virtual. El código ActionScript 3.0 puede ejecutarse con una velocidad diez veces mayor que el código ActionScript heredado.

La versión anterior de la máquina virtual ActionScript (AVM1) ejecuta código ActionScript 1.0 y ActionScript 2.0. Flash Player 9 y 10 admiten AVM9 por compatibilidad con contenido existente y heredado de versiones anteriores. Para obtener más información, consulte “Compatibilidad con versiones anteriores” en la página 7.

Novedades de ActionScript 3.0

Aunque ActionScript 3.0 contiene muchas clases y funciones que resultarán familiares a los programadores de ActionScript, la arquitectura y los conceptos de ActionScript 3.0 difieren de las versiones anteriores de ActionScript. ActionScript 3.0 incluye algunas mejoras como, por ejemplo, nuevas funciones del núcleo del lenguaje y una API de Flash Player mejorada que proporciona un mayor control de objetos de bajo nivel.

Nota: *las aplicaciones de Adobe® AIR™ también pueden utilizar las API de Flash Player.*

Funciones del núcleo del lenguaje

El núcleo del lenguaje está formado por los bloques básicos del lenguaje de programación, como sentencias, expresiones, condiciones, bucles y tipos. ActionScript 3.0 contiene muchas funciones nuevas que aceleran el proceso de desarrollo.

Excepciones de tiempo de ejecución

ActionScript 3.0 notifica más situaciones de error que las versiones anteriores de ActionScript. Las excepciones de tiempo de ejecución se utilizan en situaciones de error frecuentes y permiten mejorar la depuración y desarrollar aplicaciones para gestionar errores de forma robusta. Los errores de tiempo de ejecución pueden proporcionar trazas de pila con la información del archivo de código fuente y el número de línea. Esto permite identificar rápidamente los errores.

Tipos de tiempo de ejecución

En ActionScript 2.0, las anotaciones de tipos eran principalmente una ayuda para el desarrollador; en tiempo de ejecución, se asignaban los tipos dinámicamente a todos los valores. En ActionScript 3.0, la información de tipos se conserva en tiempo de ejecución y se utiliza con diversos fines. Flash Player y Adobe AIR realizan una verificación de tipos en tiempo de ejecución, mejorando la seguridad de los tipos del sistema. La información de tipos también se utiliza para especificar variables en representaciones nativas de la máquina, lo que mejora el rendimiento y reduce el uso de memoria.

Clases cerradas

ActionScript 3.0 introduce el concepto de clases cerradas. Una clase cerrada posee únicamente el conjunto fijo de propiedades y métodos definidos durante la compilación; no es posible añadir propiedades y métodos adicionales. Esto permite realizar una comprobación más estricta en tiempo de compilación, lo que aporta una mayor solidez a los programas. También mejora el uso de memoria, pues no requiere una tabla hash interna para cada instancia de objeto. Además, es posible utilizar clases dinámicas mediante la palabra clave `dynamic`. Todas las clases de ActionScript 3.0 están cerradas de forma predeterminada, pero pueden declararse como dinámicas con la palabra clave `dynamic`.

Cierres de métodos

ActionScript 3.0 permite que un cierre de método recuerde automáticamente su instancia de objeto original. Esta función resulta útil en la gestión de eventos. En ActionScript 2.0, los cierres de métodos no recordaban la instancia de objeto de la que se habían extraído, lo que provocaba comportamientos inesperados cuando se llamaba al cierre de método. La clase mx.utils.Delegate permitía solucionar este problema, pero ya no es necesaria.

ECMAScript for XML (E4X)
ActionScript 3.0 implementa ECMAScript for XML (E4X), recientemente estandarizado como ECMA-357. E4X ofrece un conjunto fluido y natural de construcciones del lenguaje para manipular XML. Al contrario que las API de análisis de XML tradicionales, XML con E4X se comporta como un tipo de datos nativo del lenguaje. E4X optimiza el desarrollo de aplicaciones que manipulan XML, pues reduce drásticamente la cantidad de código necesario. Para obtener más información sobre la implementación de E4X en ActionScript 3.0, consulte "Trabajo con XML" en la página 232.

Para ver la especificación de E4X publicada por ECMA, visite www.ecma-international.org.

Expresiones regulares
ActionScript 3.0 ofrece compatibilidad nativa con expresiones regulares, que permiten encontrar y manipular cadenas rápidamente. ActionScript 3.0 implementa la compatibilidad con expresiones regulares tal y como se definen en la especificación del lenguaje ECMAScript (ECMA-262) edición 3.

Espacios de nombres
Los espacios de nombres son similares a los especificadores de acceso tradicionales que se utilizan para controlar la visibilidad de las declaraciones (`public`, `private`, `protected`). Funcionan como especificadores de acceso personalizados, con nombres elegidos por el usuario. Los espacios de nombres incluyen un identificador de recursos universal (URI) para evitar colisiones y también se utilizan para representar espacios de nombres XML cuando se trabaja con E4X.

Nuevos tipos simples
ActionScript 2.0 tiene un solo tipo numérico, Number, un número de coma flotante con precisión doble. ActionScript 3.0 contiene los tipos int y uint. El tipo int es un entero de 32 bits con signo que permite al código ActionScript aprovechar las capacidades matemáticas de manipulación rápida de enteros de la CPU. Este tipo es útil para contadores de bucle y variables en las que se usan enteros. El tipo uint es un tipo entero de 32 bits sin signo que resulta útil para valores de colores RGB y recuentos de bytes, entre otras cosas.

Funciones de la API de Flash Player

Las API de Flash Player en ActionScript 3.0 contienen muchas de las clases que permiten controlar objetos a bajo nivel. La arquitectura del lenguaje está diseñada para ser mucho más intuitiva que en versiones anteriores. Hay demasiadas clases nuevas para poder tratarlas con detalle, de modo que en las siguientes secciones se destacan algunos cambios importantes.

Nota: *las aplicaciones de Adobe® AIR™ también puede utilizar las API de Flash Player.*

Modelo de eventos DOM3
El modelo de eventos del modelo de objetos de documento de nivel 3 (DOM3) ofrece un modo estándar para generar y gestionar mensajes de eventos de forma que los objetos de las aplicaciones puedan interactuar y comunicarse, mantener su estado y responder a los cambios. Diseñado a partir de la especificación de eventos DOM de nivel 3 del World Wide Web Consortium, este modelo proporciona un mecanismo más claro y eficaz que los sistemas de eventos disponibles en versiones anteriores de ActionScript.

Los eventos y los eventos de error se encuentran en el paquete flash.events. La arquitectura de componentes de Flash utiliza el mismo modelo de eventos que la API de Flash Player, de forma que el sistema de eventos está unificado en toda la plataforma Flash.

API de la lista de visualización

La API de acceso a la lista de visualización de Flash Player y Adobe AIR (el árbol que contiene todos los elementos visuales de una aplicación Flash) se compone de clases para trabajar con elementos visuales simples.

La nueva clase Sprite es un bloque básico ligero, similar a la clase MovieClip pero más apropiado como clase base de los componentes de interfaz de usuario. La nueva clase Shape representa formas vectoriales sin procesar. Es posible crear instancias de estas clases de forma natural con el operador `new` y se puede cambiar el elemento principal en cualquier momento, de forma dinámica.

La administración de profundidad es ahora automática y está incorporada en Flash Player y Adobe AIR, por lo que ya no es necesario asignar valores de profundidad. Se proporcionan nuevos métodos para especificar y administrar el orden z de los objetos.

Gestión de contenido y datos dinámicos

ActionScript 3.0 contiene mecanismos para cargar y gestionar elementos y datos en la aplicación, que son intuitivos y coherentes en toda la API. La nueva clase Loader ofrece un solo mecanismo para cargar archivos SWF y elementos de imagen, y proporciona una forma de acceso a información detallada sobre el contenido cargado. La clase URLLoader proporciona un mecanismo independiente para cargar texto y datos binarios en aplicaciones basadas en datos. La clase Socket proporciona una forma de leer y escribir datos binarios en sockets de servidor en cualquier formato.

Acceso a datos de bajo nivel

Diversas API proporcionan acceso de bajo nivel a los datos, lo que supone una novedad en ActionScript. La clase URLStream, implementada por URLLoader, proporciona acceso a los datos como datos binarios sin formato mientras se descargan. La clase ByteArray permite optimizar la lectura, escritura y utilización de datos binarios. La nueva API Sound proporciona control detallado del sonido a través de las clases SoundChannel y SoundMixer. Las nuevas API relacionadas con la seguridad proporcionan información sobre los privilegios de seguridad de un archivo SWF o contenido cargado, lo que permite gestionar mejor los errores de seguridad.

Trabajo con texto

ActionScript 3.0 contiene un paquete flash.text para todas las API relacionadas con texto. La clase TextLineMetrics proporciona medidas detalladas para una línea de texto en un campo de texto; sustituye al método `TextFormat.getTextExtent()` en ActionScript 2.0. La clase TextField contiene una serie de nuevos métodos interesantes de bajo nivel que pueden ofrecer información específica sobre una línea de texto o un solo carácter en un campo de texto. Dichos métodos son: `getCharBoundaries()`, que devuelve un rectángulo que representa el recuadro de delimitación de un carácter, `getCharIndexAtPoint()`, que devuelve el índice del carácter en un punto especificado, y `getFirstCharInParagraph()`, que devuelve el índice del primer carácter en un párrafo. Los métodos de nivel de línea son: `getLineLength()`, que devuelve el número de caracteres en una línea de texto especificada, y `getLineText()`, que devuelve el texto de la línea especificada. Una nueva clase Font proporciona un medio para administrar las fuentes incorporadas en archivos SWF.

Compatibilidad con versiones anteriores

Como siempre, Flash Player proporciona compatibilidad completa con el contenido publicado previamente con versiones anteriores. Cualquier contenido que se ejecutara en versiones anteriores de Flash Player puede ejecutarse en Flash Player 9. Sin embargo, la introducción de ActionScript 3.0 en Flash Player 9 presenta algunos retos de interoperabilidad entre el contenido antiguo y el contenido nuevo que se ejecuta en Flash Player 9. Algunos de los problemas de compatibilidad que pueden surgir son:

- No se puede combinar código ActionScript 1.0 ó 2.0 con código ActionScript 3.0 en un archivo SWF.

- El código ActionScript 3.0 puede cargar un archivo SWF escrito en ActionScript 1.0 ó 2.0, pero no puede acceder a las variables y funciones del archivo SWF.
- Los archivos SWF escritos en ActionScript 1.0 ó 2.0 no pueden cargar archivos SWF escritos en ActionScript 3.0. Esto significa que los archivos SWF creados en Flash 8 o Flex Builder 1.5 o versiones anteriores no pueden cargar archivos SWF de ActionScript 3.0.

  La única excepción a esta regla es que un archivo SWF de ActionScript 2.0 puede sustituirse a sí mismo por un archivo SWF de ActionScript 3.0, siempre y cuando el archivo SWF de ActionScript 2.0 no haya cargado ningún elemento en ninguno de sus niveles. Para ello, el archivo SWF de ActionScript 2.0 debe realizar una llamada a `loadMovieNum()`, pasando un valor 0 al parámetro `level`.
- En general, los archivos SWF escritos en ActionScript 1.0 ó 2.0 se deben migrar si van a funcionar de forma conjunta con los archivos SWF escritos en ActionScript 3.0. Por ejemplo, supongamos que se ha creado un reproductor de medios utilizando ActionScript 2.0. El reproductor carga distinto contenido que también se creó utilizando ActionScript 2.0. No es posible crear nuevo contenido en ActionScript 3.0 y cargarlo en el reproductor de medios. Es necesario migrar el reproductor de vídeo a ActionScript 3.0.

  No obstante, si se crea un reproductor de medios en ActionScript 3.0, dicho reproductor puede realizar cargas sencillas del contenido de ActionScript 2.0.

En la siguiente tabla se resumen las limitaciones de las versiones anteriores de Flash Player en lo referente a la carga de nuevo contenido y a la ejecución de código, así como las limitaciones relativas a la reutilización de scripts entre archivos SWF escritos en distintas versiones de ActionScript.

Funcionalidad admitida	**Flash Player 7**	**Flash Player 8**	**Flash Player 9 y 10**
Puede cargar archivos SWF publicados para	7 y versiones anteriores	8 y versiones anteriores	9 (o 10) y versiones anteriores
Contiene esta AVM	AVM1	AVM1	AVM1 y AVM2
Ejecuta archivos SWF escritos en ActionScript	1.0 y 2.0	1.0 y 2.0	1.0, 2.0 y 3.0

En la siguiente tabla, “Funcionalidad admitida” hace referencia al contenido que se ejecuta en Flash Player 9 o posterior. El contenido ejecutado en Flash Player 8 o versiones anteriores puede cargar, mostrar, ejecutar y reutilizar scripts únicamente de ActionScript 1.0 y 2.0.

Funcionalidad admitida	**Contenido creado en ActionScript 1.0 y 2.0**	**Contenido creado en ActionScript 3.0**
Puede cargar contenido y ejecutar código en contenido creado en	Sólo ActionScript 1.0 y 2.0	ActionScript 1.0 y 2.0, y ActionScript 3.0
Puede reutilizar contenido de scripts creado en	Sólo ActionScript 1.0 y 2.0 (ActionScript 3.0 a través de conexión local)	ActionScript 1.0 y 2.0 a través de LocalConnection. ActionScript 3.0

Capítulo 3: Introducción a ActionScript

Este capítulo se ha diseñado como punto de partida para empezar a programar en ActionScript. Aquí se proporcionan las bases necesarias para comprender los conceptos y ejemplos descritos en el resto de páginas de este manual. Para comenzar, se ofrece una descripción de los conceptos básicos de programación en el contexto de su aplicación en ActionScript. También se tratan los aspectos fundamentales de la organización y creación de una aplicación ActionScript.

Fundamentos de programación

Dado que ActionScript es un lenguaje de programación, será de gran ayuda comprender primero algunos conceptos generales de programación de ordenadores.

Para qué sirven los programas informáticos

En primer lugar, resulta útil entender qué es un programa informático y para qué sirve. Un programa informático se caracteriza por dos aspectos principales:

- Un programa es una serie de instrucciones o pasos que debe llevar a cabo el equipo.
- Cada paso implica en última instancia la manipulación de información o datos.

En general, un programa informático es simplemente una lista de instrucciones paso a paso que se dan al equipo para que las lleve a cabo una a una. Cada una de las instrucciones se denomina *sentencia*. Como se verá a lo largo de este manual, en ActionScript cada sentencia finaliza con un punto y coma.

Lo que realiza básicamente una instrucción dada en un programa es manipular algún bit de datos almacenado en la memoria del equipo. En un caso sencillo, se puede indicar al equipo que sume dos números y almacene el resultado en su memoria. En un caso más complejo, se podría tener un rectángulo dibujado en la pantalla y escribir un programa para moverlo a algún otro lugar de la pantalla. El equipo realiza un seguimiento de determinada información relativa al rectángulo: las coordenadas x e y que indican su ubicación, la anchura y altura, el color, etc. Cada uno de estos bits de información se almacena en algún lugar de la memoria del equipo. Un programa para mover el rectángulo a otra ubicación incluiría pasos como "cambiar la coordenada x a 200; cambiar la coordenada y a 150" (especificando nuevos valores para las coordenadas x e y). Por supuesto, el equipo procesa estos datos de algún modo para convertir estos números en la imagen que aparece en la pantalla; pero para el nivel de detalle que aquí interesa, basta con saber que el proceso de "mover un rectángulo en la pantalla" sólo implica en realidad un cambio de bits de datos en la memoria del equipo.

Variables y constantes

Dado que la programación implica principalmente cambiar datos en la memoria del equipo, tiene que haber una forma de representar un solo dato en el programa. Una *variable* es un nombre que representa un valor en la memoria del equipo. Cuando se escriben sentencias para manipular valores, se escribe el nombre de la variable en lugar del valor; cuando el equipo ve el nombre de la variable en el programa, busca en su memoria y utiliza el valor que allí encuentra. Por ejemplo, si hay dos variables denominadas `value1` y `value2`, cada una de las cuales contiene un número, para sumar esos dos números se podría escribir la sentencia:

```
value1 + value2
```

Cuando lleve a cabo los pasos indicados, el equipo buscará los valores de cada variable y los sumará.

En ActionScript 3.0, una variable se compone realmente de tres partes distintas:

- El nombre de la variable
- El tipo de datos que puede almacenarse en la variable
- El valor real almacenado en la memoria del equipo

Se acaba de explicar cómo el equipo utiliza el nombre como marcador de posición del valor. El tipo de datos también es importante. Cuando se crea una variable en ActionScript, se especifica el tipo concreto de datos que contendrá; a partir de ahí, las instrucciones del programa sólo pueden almacenar ese tipo de datos en la variable y se puede manipular el valor con las características particulares asociadas a su tipo de datos. En ActionScript, para crear una variable (se conoce como *declarar* la variable), se utiliza la sentencia `var`:

```
var value1:Number;
```

En este caso, se ha indicado al equipo que cree una variable denominada `value1`, que contendrá únicamente datos numéricos ("Number" es un tipo de datos específico definido en ActionScript). También es posible almacenar un valor directamente en la variable:

```
var value2:Number = 17;
```

En Adobe Flash CS4 Professional hay otra forma posible de declarar una variable. Cuando se coloca un símbolo de clip de película, un símbolo de botón o un campo de texto en el escenario, se le puede asignar un nombre de instancia en el inspector de propiedades. En segundo plano, Flash crea una variable con el mismo nombre que la instancia, que se puede utilizar en el código ActionScript para hacer referencia a ese elemento del escenario. Así, por ejemplo, si hay un símbolo de clip de película en el escenario y se le asigna el nombre de instancia`rocketShip`, siempre que se use la variable `rocketShip` en el código ActionScript, se estará manipulando dicho clip de película.

Una constante es muy similar a una variable en el sentido de que es un nombre que representa a un valor en la memoria del equipo, con un tipo de datos específico. La diferencia es que a una constante sólo se le puede asignar un valor cada vez en el curso de una aplicación ActionScript. Tras asignar un valor a una constante, éste permanecerá invariable en toda la aplicación. La sintaxis para declarar constantes coincide con la de las variables, excepto por el hecho de que se usa la palabra clave `const` en lugar de `var`:

```
const SALES_TAX_RATE:Number = 0.07;
```

Una constante resulta útil para definir un valor que se utilizará en varios puntos de un proyecto y que no cambiará en circunstancias normales. Cuando se utiliza una constante en lugar de un valor literal el código resulta más legible. Por ejemplo, es más fácil entender la finalidad de una línea de código que multiplica un precio por `SALES_TAX_RATE` que la de una línea de código que lo haga por `0.07`. Además, si en un momento dado es preciso cambiar el valor definido por una constante, sólo habrá que hacerlo en un punto (la declaración de la constante) cuando se utiliza una constante para su representación en el proyecto, en lugar de tener que modificarlo varias veces como cuando se utilizan valores literales especificados en el código.

Tipos de datos

En ActionScript, hay muchos tipos de datos que pueden utilizarse como el tipo de datos de las variables que se crean. Algunos de estos tipos de datos se pueden considerar "sencillos" o "fundamentales":

- String: un valor de texto como, por ejemplo, un nombre o el texto de un capítulo de un libro
- Numeric: ActionScript 3.0 incluye tres tipos de datos específicos para datos numéricos:
  - Number: cualquier valor numérico, incluidos los valores fraccionarios o no fraccionarios
  - int: un entero (un número no fraccionario)
  - uint: un entero sin signo, es decir, que no puede ser negativo

- Boolean: un valor true (verdadero) o false (falso), por ejemplo, si un conmutador está activado o si dos valores son iguales

El tipo de datos sencillo representa a un solo elemento de información: por ejemplo, un único número o una sola secuencia de texto. No obstante, la mayoría de los tipos de datos definidos en ActionScript podrían describirse como tipos de datos complejos porque representan un conjunto de valores agrupados. Por ejemplo, una variable con el tipo de datos Date representa un solo valor: un momento temporal. No obstante, ese valor de fecha se representa en realidad en forma de diferentes valores: el día, el mes, el año, las horas, los minutos, los segundos, etc., los cuales son números individuales. Así pues, aunque se perciba una fecha como un solo valor (y se pueda tratar como tal creando una variable Date), internamente el equipo lo considera un grupo de varios valores que conjuntamente definen una sola fecha.

La mayoría de los tipos de datos incorporados y los tipos de datos definidos por los programadores son complejos. Algunos de los tipos de datos complejos que podrían reconocerse son:

- MovieClip: un símbolo de clip de película
- TextField: un campo de texto dinámico o de texto de entrada
- SimpleButton: un símbolo de botón
- Date: información sobre un solo momento temporal (una fecha y hora)

Para referirse a los tipos de datos, a menudo se emplean como sinónimos las palabras clase y objeto. Una *clase* es simplemente la definición de un tipo de datos — es una especie de plantilla para todos los objetos del tipo de datos, como afirmar que "todas las variables del tipo de datos Example tiene estas características: A, B y C". Un *objeto*, por otra parte, es una instancia real de una clase; una variable cuyo tipo de datos sea MovieClip se podría describir como un objeto MovieClip. Se puede decir lo mismo con distintos enunciados:

- El tipo de datos de la variable `myVariable` es Number.
- La variable `myVariable` es una instancia de Number.
- La variable `myVariable` es un objeto Number.
- La variable `myVariable` es una instancia de la clase Number.

Trabajo con objetos

ActionScript es lo que se denomina un lenguaje de programación orientado a objetos. La programación orientada a objetos es simplemente un enfoque de la programación, es decir, una forma de organizar el código en un programa mediante objetos.

Anteriormente se ha definido un programa informático como una serie de pasos o instrucciones que lleva a cabo el equipo. Así pues, conceptualmente se podría imaginar un programa informático simplemente como una larga lista de instrucciones. Sin embargo, en la programación orientada a objetos, las instrucciones del programa se dividen entre distintos objetos; el código se agrupa en segmentos de funcionalidad, de modo que los tipos de funcionalidad relacionados o los elementos de información relacionados se agrupan en un contenedor.

De hecho, si se ha trabajado con símbolos en Flash, se estará acostumbrado a trabajar con objetos. Supongamos que se ha definido un símbolo de clip de película (por ejemplo, el dibujo de un rectángulo) y se ha colocado una copia del mismo en el escenario. Dicho símbolo de clip de película también es (literalmente) un objeto en ActionScript; es una instancia de la clase MovieClip.

Es posible modificar algunas de las características del clip de película. Por ejemplo, cuando está seleccionado, es posible cambiar algunos valores en el inspector de propiedades como, por ejemplo, la coordenada x o la anchura, o realizar algunos ajustes de color como cambiar su valor de transparencia alfa o aplicarle un filtro de sombra. Otras herramientas de Flash permiten realizar más cambios, como utilizar la herramienta Transformación libre para girar el rectángulo. Todas estas acciones para modificar un símbolo de clip de película en el entorno de edición de Flash también se pueden realizar en ActionScript cambiando los elementos de datos que se agrupan en un único paquete denominado objeto MovieClip.

En la programación orientada a objetos de ActionScript, hay tres tipos de características que puede contener cualquier clase:

- Propiedades
- Métodos
- Eventos

Estos elementos se utilizan conjuntamente para administrar los elementos de datos que utiliza el programa y para decidir qué acciones deben llevarse a cabo y en qué orden.

Propiedades

Una propiedad representa uno de los elementos de datos que se empaquetan en un objeto. Un objeto Song (canción) puede tener propiedades denominadas `artist` (artista) y `title` (título); la clase MovieClip tiene propiedades como `rotation` (rotación), `x`, `width` (anchura) y `alpha` (alfa). Se trabaja con las propiedades del mismo modo que con las variables individuales; de hecho, se podría pensar que las propiedades son simplemente las variables "secundarias" contenidas en un objeto.

A continuación se muestran algunos ejemplos de código ActionScript que utiliza propiedades. Esta línea de código mueve el objeto MovieClip denominado `square` a la coordenada x = 100 píxeles:

```
square.x = 100;
```

Este código utiliza la propiedad rotation para que el MovieClip `square` gire de forma correspondiente a la rotación del MovieClip `triangle`:

```
square.rotation = triangle.rotation;
```

Este código altera la escala horizontal del MovieClip `square` para hacerlo 1,5 veces más ancho:

```
square.scaleX = 1.5;
```

Fíjese en la estructura común: se utiliza una variable (`square`, `triangle`) como nombre del objeto, seguida de un punto (`.`) y, a continuación, el nombre de la propiedad (`x`, `rotation`, `scaleX`). El punto, denominado *operador de punto*, se utiliza para indicar el acceso a uno de los elementos secundarios de un objeto. El conjunto de la estructura (nombre de variable-punto-nombre de propiedad) se utiliza como una sola variable, como un nombre de un solo valor en la memoria del equipo.

Métodos

Un *método* es una acción que puede llevar a cabo un objeto. Por ejemplo, si se ha creado un símbolo de clip de película en Flash con varios fotogramas clave y animación en la línea de tiempo, ese clip de película podrá reproducirse, detenerse o recibir instrucciones para mover la cabeza lectora a un determinado fotograma.

Este código indica al objeto MovieClip denominado `shortFilm` que inicie su reproducción:

```
shortFilm.play();
```

Esta línea hace que el MovieClip denominado `shortFilm` deje de reproducirse (la cabeza lectora se detiene como si se hiciera una pausa en un vídeo):

```
shortFilm.stop();
```

Este código hace que un MovieClip denominado `shortFilm` mueva su cabeza lectora al fotograma 1 y deje de reproducirse (como si se rebobinara un vídeo):

```
shortFilm.gotoAndStop(1);
```

Como puede verse, para acceder a los métodos, se debe escribir el nombre del objeto (una variable), un punto y el nombre del método seguido de un paréntesis, siguiendo la misma estructura que para las propiedades. El paréntesis es una forma de indicar que se está *llamando* al método, es decir, indicando al objeto que realice esa acción. Algunos valores (o variables) se incluyen dentro del paréntesis para pasar información adicional necesaria para llevar a cabo la acción. Estos valores se denominan *parámetros* del método. Por ejemplo, el método `gotoAndStop()` necesita saber cuál es el fotograma al que debe dirigirse, de modo que requiere un solo parámetro en el paréntesis. Otros métodos como `play()` y `stop()` no requieren información adicional porque son descriptivos por sí mismos. Sin embargo, también se escriben con paréntesis.

A diferencia de las propiedades (y las variables), los métodos no se usan como identificadores de valores. No obstante, algunos métodos pueden realizar cálculos y devolver un resultado que puede usarse como una variable. Por ejemplo, el método `toString()` de la clase Number convierte el valor numérico en su representación de texto:

```
var numericData:Number = 9;
var textData:String = numericData.toString();
```

Por ejemplo, se usaría el método `toString()` para mostrar el valor de una variable Number en un campo de texto de la pantalla. La propiedad `text` de la clase TextField (que representa el contenido de texto real que se muestra en la pantalla) se define como String (cadena), de modo que sólo puede contener valores de texto. Esta línea de código convierte en texto el valor numérico de la variable `numericData` y, a continuación, hace que aparezca en la pantalla en el objeto TextField denominado `calculatorDisplay`:

```
calculatorDisplay.text = numericData.toString();
```

Eventos

Se ha descrito un programa informático como una serie de instrucciones que el ordenador lleva a cabo paso a paso. Algunos programas informáticos sencillos no son más que eso: unos cuantos pasos que el ordenador ejecuta, tras los cuales finaliza el programa. Sin embargo, los programas de ActionScript se han diseñado para continuar ejecutándose, esperando los datos introducidos por el usuario u otras acciones. Los eventos son los mecanismos que determinan qué instrucciones lleva a cabo el ordenador y cuándo las realiza.

Básicamente, los *eventos* son acciones que ActionScript conoce y a las que puede responder. Muchos eventos se relacionan con la interacción del usuario (hacer clic en un botón, presionar una tecla del teclado, etc.) pero también existen otros tipos de eventos. Por ejemplo, si se usa ActionScript para cargar una imagen externa, existe un evento que puede indicar al usuario cuándo finaliza la carga de la imagen. En esencia, cuando se ejecuta un programa de ActionScript, Adobe Flash Player y Adobe AIR simplemente esperan a que ocurran determinadas acciones y, cuando suceden, ejecutan el código ActionScript que se haya especificado para tales eventos.

Gestión básica de eventos

La técnica para especificar determinadas acciones que deben realizarse como respuesta a eventos concretos se denomina *gestión de eventos*. Cuando se escribe código ActionScript para llevar a cabo la gestión de eventos, se deben identificar tres elementos importantes:

- El origen del evento: ¿en qué objeto va a repercutir el evento? Por ejemplo, ¿en qué botón se hará clic o qué objeto Loader está cargando la imagen? El origen del evento también se denomina *objetivo del evento*, ya que es el objeto al que Flash Player o AIR destinan el evento (es decir, donde éste tiene lugar realmente).
- El evento: ¿qué va a suceder, a qué se va a responder? Es importante identificar esto porque muchos objetos activan varios eventos.
- La respuesta: ¿qué pasos hay que llevar a cabo cuando ocurra el evento?

Siempre que se escriba código ActionScript para gestionar eventos, el código debe incluir estos tres elementos y debe seguir esta estructura básica (los elementos en negrita son marcadores de posición que hay que completar en cada caso concreto):

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

eventSource.addEventListener(EventType.EVENT_NAME, eventResponse);
```

Este código realiza dos acciones. En primer lugar, define una función, que es la forma de especificar las acciones que desean realizarse como respuesta al evento. A continuación, llama al método `addEventListener()` del objeto de origen, básicamente "suscribiendo" la función al evento especificado de modo que se lleven a cabo las acciones de la función cuando ocurra el evento. Cada una de estas partes se tratará con mayor detalle.

Una *función* proporciona un modo de agrupar acciones con un único nombre que viene a ser un nombre de método abreviado para llevar a cabo las acciones. Una función es idéntica a un método excepto en que no está necesariamente asociada a una clase determinada (de hecho, es posible definir un método como una función asociada a una clase determinada). Cuando se crea una función para la gestión de eventos, se debe elegir el nombre de la función (denominada `eventResponse` en este caso). Además, se debe especificar un parámetro (denominado `eventObject` en este ejemplo). Especificar un parámetro de una función equivale a declarar una variable, de modo que también hay que indicar el tipo de datos del parámetro. (En este ejemplo, el tipo de datos del parámetro es `EventType`.)

Cada tipo de evento que se desee detectar tiene asociada una clase de ActionScript. El tipo de datos especificado para el parámetro de función es siempre la clase asociada del evento concreto al que se desea responder. Por ejemplo, un evento `click` (el cual se activa al hacer clic en un elemento con el ratón) se asocia a la clase MouseEvent. Cuando se vaya a escribir una función de detector para un evento `click`, ésta se debe definir con un parámetro con el tipo de datos MouseEvent. Por último, entre las llaves de apertura y cierre ({ ... }), se escriben las instrucciones que debe llevar a cabo el equipo cuando ocurra el evento.

Después de escribir la función de gestión de eventos, es necesario indicar al objeto de origen del evento (el objeto en el que se produce el evento, por ejemplo, el botón) que se desea llamar a la función cuando ocurra el evento. Para ello es necesario llamar al método `addEventListener()` de dicho objeto (todos los objetos que tienen eventos también tienen un método `addEventListener()`). El método `addEventListener()` utiliza dos parámetros:

- En primer lugar, el nombre del evento específico al que se desea responder. De nuevo, cada evento se asocia a una clase específica, que tiene a su vez un valor especial predefinido para cada evento (como un nombre exclusivo propio del evento), que debe usarse como primer parámetro.
- En segundo lugar, el nombre de la función de respuesta al evento. Hay que tener en cuenta que el nombre de una función debe escribirse sin paréntesis cuando se pasa como un parámetro.

Análisis del proceso de gestión de eventos

A continuación se ofrece una descripción paso a paso del proceso que tiene lugar cuando se crea un detector de eventos. En este caso, es un ejemplo de creación de función de detector a la que se llama cuando se hace clic en un objeto denominado `myButton`.

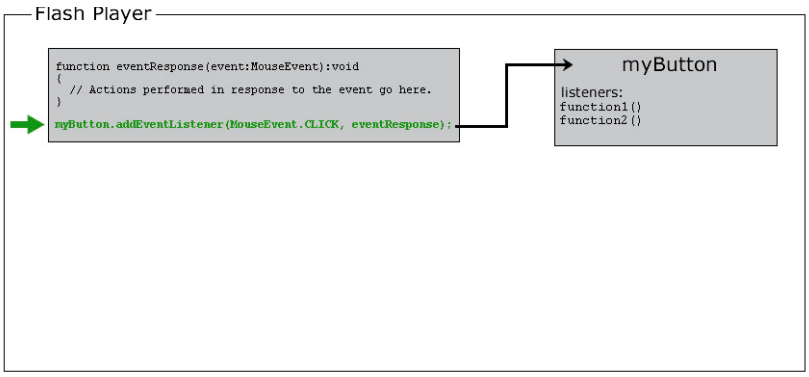
El código escrito por el programador es el siguiente:

```
function eventResponse(event:MouseEvent):void
{
    // Actions performed in response to the event go here.
}

myButton.addEventListener(MouseEvent.CLICK, eventResponse);
```

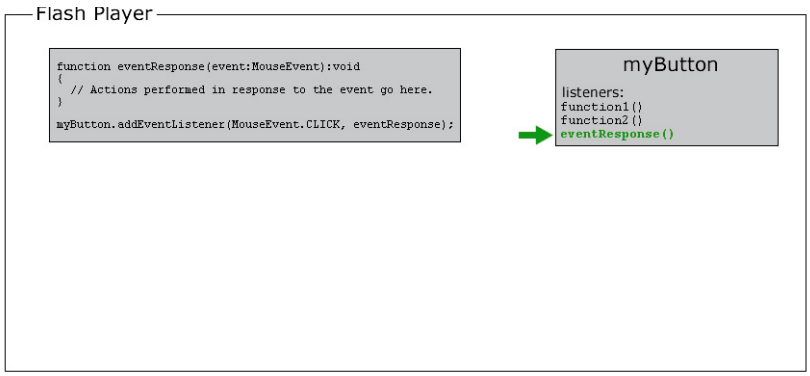
Al ejecutarse en Flash Player, el código funcionaría de la manera siguiente. (El comportamiento es idéntico en Adobe AIR):

1 Cuando se carga el archivo SWF, Flash Player detecta que existe una función denominada `eventResponse()`.

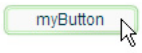
2 A continuación, Flash Player ejecuta el código (específicamente, las líneas de código que no están en una función). En este caso se trata de una sola línea de código: para llamar al método `addEventListener()` en el objeto de origen del evento (denominado `myButton`) y pasar la función `eventResponse` como parámetro.



a Internamente, `myButton` tiene una lista de funciones que detecta cada uno de sus eventos, por lo que cuando se llama a su método `addEventListener()`, `myButton` almacena la función `eventResponse()` en su lista de detectores de eventos.

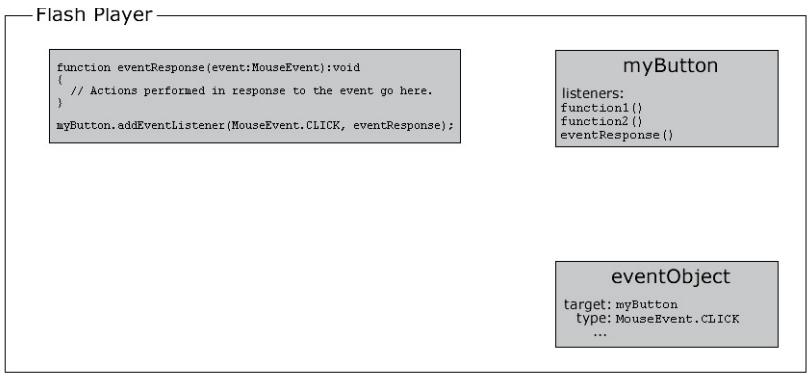


3 Cuando el usuario hace clic en el objeto `myButton`, se activa el evento `click` (identificado como `MouseEvent.CLICK` en el código).

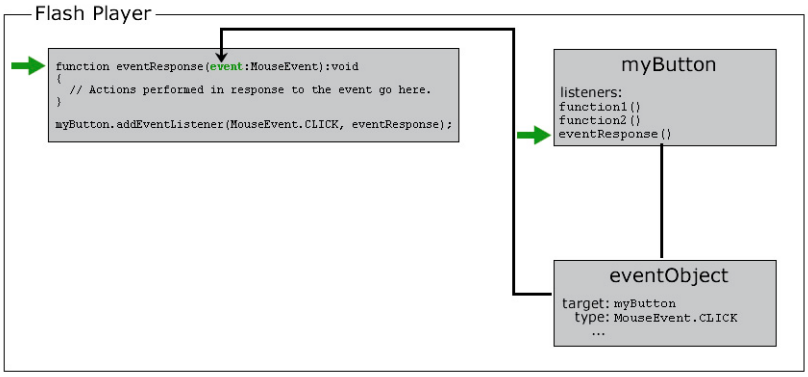


En este punto ocurre lo siguiente:

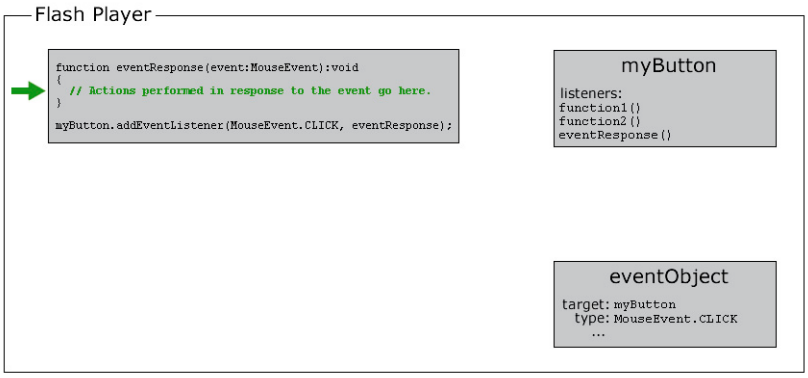
a Flash Player crea un objeto, una instancia de la clase asociada con el evento en cuestión (MouseEvent en este ejemplo). Para muchos eventos esto será una instancia de la clase Event, para eventos del ratón será una instancia de MouseEvent y para otros eventos será una instancia de la clase asociada con el evento. Este objeto creado se conoce como el *objeto de evento* y contiene información específica sobre el evento que se ha producido: el tipo de evento, el momento en que ha ocurrido y otros datos relacionados con el evento, si procede.



b A continuación, Flash Player busca en la lista de detectores de eventos almacenada en `myButton`. Recorre estas funciones de una en una, llamando a cada función y pasando el objeto de evento a la función como parámetro. Como la función `eventResponse()` es uno de los detectores de `myButton`, como parte de este proceso Flash Player llama a la función `eventResponse()`.



c Cuando se llama a la función `eventResponse()`, se ejecuta el código de la función para realizar las acciones especificadas.



Ejemplos de gestión de eventos

A continuación se ofrecen más ejemplos concretos de eventos para proporcionar una idea de algunos de los elementos comunes de los eventos y de las posibles variaciones disponibles cuando se escribe código de gestión de eventos:

- Hacer clic en un botón para iniciar la reproducción del clip de película actual. En el siguiente ejemplo, `playButton` es el nombre de instancia del botón y `this` es el nombre especial, que significa "el objeto actual":

```
this.stop();

function playMovie(event:MouseEvent):void
{
    this.play();
}

playButton.addEventListener(MouseEvent.CLICK, playMovie);
```

- Detectar si se ha escrito algo en un campo de texto. En este ejemplo, `entryText` es un campo de introducción de texto y `outputText` es un campo de texto dinámico:

```
function updateOutput(event:TextEvent):void
{
    var pressedKey:String = event.text;
    outputText.text = "You typed: " + pressedKey;
}

entryText.addEventListener(TextEvent.TEXT_INPUT, updateOutput);
```

- Hacer clic en un botón para navegar a un URL. En este caso, `linkButton` es el nombre de instancia del botón:

```
function gotoAdobeSite(event:MouseEvent):void
{
    var adobeURL:URLRequest = new URLRequest("http://www.adobe.com/");
    navigateToURL(adobeURL);
}

linkButton.addEventListener(MouseEvent.CLICK, gotoAdobeSite);
```

Creación de instancias de objetos

Por supuesto, antes de poder utilizar un objeto en ActionScript, éste debe existir. Una parte de la creación de un objeto es la declaración de una variable; sin embargo, declarar una variable sólo crea un espacio vacío en la memoria del equipo. Es necesario asignar un valor real a la variable, es decir, crear un objeto y almacenarlo en la variable, antes de intentar usarla o manipularla. El proceso de creación de un objeto se denomina *creación de una instancia* del objeto, en concreto, de una clase determinada.

Hay una forma sencilla de crear una instancia de objeto en la que no se utiliza ActionScript en absoluto. En Flash, cuando se coloca un símbolo de clip de película, un símbolo de botón o un campo de texto en el escenario, y se le asigna un nombre de instancia en el inspector de propiedades, Flash declara automáticamente una variable con ese nombre de instancia, crea una instancia de objeto y almacena el objeto en la variable. Del mismo modo, en Adobe Flex Builder, cuando se crea un componente en MXML (codificando una etiqueta MXML o colocando el componente en el editor en modo de diseño) y se le asigna un ID (en el formato MXML o en la vista Propiedades de Flex), ese ID se convierte en el nombre de una variable de ActionScript y se crea una instancia del componente, que se almacena en la variable.

Sin embargo, no siempre se desea crear un objeto visualmente. Hay varias formas de crear instancias de objetos utilizando exclusivamente ActionScript. En primer lugar, con varios tipos de datos de ActionScript, es posible crear una instancia con una *expresión literal*, es decir, un valor escrito directamente en el código ActionScript. A continuación se muestran algunos ejemplos:

- Valor numérico literal (introducir el número directamente):

```
var someNumber:Number = 17.239;
var someNegativeInteger:int = -53;
var someUint:uint = 22;
```

- Valor de cadena literal (poner el texto entre comillas dobles):

```
var firstName:String = "George";
var soliloquy:String = "To be or not to be, that is the question...";
```

- Valor booleano literal (usar los valores literales `true` o `false`):

```
var niceWeather:Boolean = true;
var playingOutside:Boolean = false;
```

- Valor de conjunto literal (cerrar entre corchetes una lista de valores separados por coma):

```
var seasons:Array = ["spring", "summer", "autumn", "winter"];
```

- Valor XML literal (introducir los datos XML directamente):

```
var employee:XML = <employee>
        <firstName>Harold</firstName>
        <lastName>Webster</lastName>
    </employee>;
```

ActionScript también define expresiones literales para los tipos de datos Array, RegExp, Object y Function. Para obtener información detallada sobre estas clases, consulte "Trabajo con conjuntos" en la página 159, "Utilización de expresiones regulares" en la página 211 y "Tipo de datos Object" en la página 61.

En los demás tipos de datos, para crear una instancia de objeto, se utiliza el operador `new` con el nombre de clase, como en el siguiente ejemplo:

```
var raceCar:MovieClip = new MovieClip();
var birthday:Date = new Date(2006, 7, 9);
```

A menudo se hace referencia a la creación de un objeto con el operador `new` como "llamar al constructor de la clase". Un *constructor* es un método especial al que se llama como parte del proceso de creación de una instancia de una clase. Debe tenerse en cuenta que, cuando se crea una instancia de este modo, se ponen paréntesis después del nombre de la clase y en ocasiones se especifican los valores del parámetro, dos cosas que también se realizan cuando se llama a un método.

Tenga en cuenta que, incluso en los tipos de datos que permiten crear instancias con una expresión literal, se puede utilizar el operador `new` para crear una instancia de objeto. Por ejemplo, las siguientes dos líneas de código realizan exactamente lo mismo:

```
var someNumber:Number = 6.33;
var someNumber:Number = new Number(6.33);
```

Es importante familiarizarse con la creación de objetos mediante `new` *ClassName*`()`. Si se necesita crear una instancia de cualquier tipo de datos de ActionScript que no tenga una representación visual (y que, por lo tanto, no pueda crearse colocando un elemento en el escenario de Flash o el modo de diseño del editor MXML de Flex Builder), sólo puede hacerse creando el objeto directamente en ActionScript con el operador `new`.

En Flash concretamente, el operador `new` también se puede usar para crear una instancia de un símbolo de clip de película que esté definido en la biblioteca pero no esté colocado en el escenario. Para obtener más información al respecto, consulte "Creación de objetos MovieClip con ActionScript" en la página 422.

Elementos comunes de los programas

Además de la declaración de variables, la creación de instancias de objetos y la manipulación de objetos mediante sus propiedades y métodos, hay otros bloques de creación que se pueden usar para crear un programa de ActionScript.

Operadores

Los *operadores* son símbolos especiales (o, en ocasiones, palabras) que se utilizan para realizar cálculos. Se utilizan principalmente en las operaciones matemáticas, pero también en la comparación entre valores. Por lo general, un operador utiliza uno o varios valores, y calcula un solo resultado. Por ejemplo:

- El operador de suma (`+`) suma dos valores y obtiene como resultado una sola cifra:

  ```
  var sum:Number = 23 + 32;
  ```

- El operador de multiplicación (`*`) multiplica un valor por otro y obtiene como resultado una sola cifra:

  ```
  var energy:Number = mass * speedOfLight * speedOfLight;
  ```

- El operador de igualdad (`==`) compara dos valores para ver si son iguales y obtiene como resultado un solo valor booleano (true o false):

  ```
  if (dayOfWeek == "Wednesday")
  {
      takeOutTrash();
  }
  ```

  Como se muestra aquí, el operador de igualdad y los otros operadores de comparación se suelen utilizar con la sentencia `if` para determinar si determinadas instrucciones deben llevarse a cabo o no.

Para ver más detalles y ejemplos sobre el uso de los operadores, consulte "Operadores" en la página 71.

Comentarios

Mientras se escribe código ActionScript, a menudo se desea dejar notas para uno mismo, quizás para explicar el funcionamiento de algunas líneas de código o el motivo de una determinada elección. Los *comentarios del código* son una herramienta que permite escribir en el código texto que el ordenador debe ignorar. ActionScript incluye dos tipos de comentarios:

- Comentario de una sola línea: un comentario de una sola línea se indica mediante dos barras diagonales en cualquier lugar de una línea. El ordenador omitirá el texto entre las dos barras y el final de la línea:

```
// This is a comment; it's ignored by the computer.
var age:Number = 10; // Set the age to 10 by default.
```

- Comentario multilínea: un comentario multilínea contiene un marcador de inicio del comentario (`/*`), el contenido del comentario y un marcador de fin del comentario (`*/`). Todo el texto entre los marcadores de inicio y de fin será omitido por el ordenador, independientemente del número de líneas que ocupe el comentario:

```
/*
This might be a really long description, perhaps describing what
a particular function is used for or explaining a section of code.

In any case, these lines are all ignored by the computer.
*/
```

Los comentarios también se utilizan con frecuencia para "desactivar" una o varias líneas de código. Por ejemplo, si se está probando una forma distinta de llevar a cabo algo o se está intentando saber por qué determinado código ActionScript no funciona del modo esperado.

Control de flujo

En un programa, muchas veces se desea repetir determinadas acciones, realizar sólo algunas acciones y no otras, o realizar acciones alternativas en función de determinadas condiciones, etc. El *control de flujo* es el control sobre el cual se llevan a cabo las funciones. Hay varios tipos de elementos de control de flujo disponibles en ActionScript.

- Funciones: las funciones son como los métodos abreviados; proporcionan un modo de agrupar una serie de acciones bajo un solo nombre y pueden utilizarse para realizar cálculos. Las funciones son especialmente importantes en la gestión de eventos, pero también se utilizan como una herramienta general para agrupar una serie de instrucciones. Para obtener más información sobre funciones, consulte “Funciones” en la página 81.
- Bucles: las estructuras de bucle permiten designar una serie de instrucciones que el equipo realizará un número definido de veces o hasta que cambie alguna condición. A menudo los bucles se utilizan para manipular varios elementos relacionados, mediante una variable cuyo valor cambia cada vez que el ordenador recorre el bucle. Para obtener más información sobre bucles, consulte “Reproducir indefinidamente” en la página 79.
- Sentencias condicionales: las sentencias condicionales proporcionan un modo de designar determinadas instrucciones que sólo se llevan a cabo bajo circunstancias concretas o de ofrecer conjuntos alternativos de instrucciones para condiciones distintas. El tipo más común de sentencia condicional es la sentencia `if`. La sentencia `if` comprueba un valor o una expresión escrita entre paréntesis. Si el valor es `true`, se ejecutan las líneas de código entre llaves; de lo contrario, se omiten. Por ejemplo:

```
if (age < 20)
{
    // show special teenager-targeted content
}
```

  La pareja de la sentencia `if`, la sentencia `else`, permite designar instrucciones alternativas que se llevarán a cabo si la condición no es `true`:

```
if (username == "admin")
{
    // do some administrator-only things, like showing extra options
}
else
{
    // do some non-administrator things
}
```

Para obtener más información sobre sentencias condicionales, consulte “Condicionales” en la página 77.

Ejemplo: Sitio de muestras de animación

Este ejemplo se ha diseñado para ofrecer una primera oportunidad de consultar cómo se pueden juntar fragmentos de ActionScript para crear una aplicación completa llena de código ActionScript. El ejemplo del sitio de muestras de animación es un ejemplo de cómo se puede partir de una animación lineal (por ejemplo, un trabajo creado para un cliente) y añadir algunos elementos interactivos secundarios adecuados para incorporar la animación en un sitio de muestras en línea. El comportamiento interactivo que añadiremos a la animación incluirá dos botones en los que el espectador podrá hacer clic: uno para iniciar la animación y otro para navegar a un URL diferente (como el menú del sitio o la página principal del autor).

El proceso de crear este trabajo puede dividirse en las siguientes partes principales:

1. Preparar el archivo FLA para añadir código ActionScript y elementos interactivos.
2. Crear y añadir los botones.
3. Escribir el código ActionScript.
4. Probar la aplicación.

Preparación de la animación para añadirle interactividad

Para poder añadir elementos interactivos a la animación, es útil configurar el archivo FLA creando algunos lugares para añadir el contenido nuevo. Esto incluye la creación en el escenario del espacio en el que se colocarán los botones y la creación de "espacio" en el archivo FLA para mantener separados los distintos elementos.

Para preparar el archivo FLA para añadir elementos interactivos:

1. Si no dispone de una animación lineal a la que añadir interactividad, cree un nuevo archivo FLA con una animación sencilla, como una interpolación de movimiento o de forma individual. Si no, abra el archivo FLA que contiene la animación que va a exhibir en el proyecto y guárdela con un nuevo nombre para crear un nuevo archivo de trabajo.
2. Decida en qué parte de la pantalla van a aparecer los dos botones (uno para iniciar la animación y otro vinculado al sitio de muestras o a la página principal del autor). Si es necesario, borre o añada espacio en el escenario para el nuevo contenido. Si la animación no incluye una pantalla de bienvenida, puede que desee crear una en el primer fotograma (probablemente tenga que desplazar la animación de forma que empiece en el Fotograma 2 o en un fotograma posterior).
3. Añada una nueva capa sobre las otras capas de la línea de tiempo y asigne el nombre **buttons**. Ésta será la capa a la que añadirá los botones.
4. Añada otra capa sobre la capa buttons y asígnele el nombre **actions**. Ésta será la capa en la que añadirá el código ActionScript para la aplicación.

Crear y añadir botones

A continuación hay que crear y colocar los botones que forman el centro de la aplicación interactiva.

Para crear y añadir botones al archivo FLA:

1. Utilice las herramientas de dibujo para crear el aspecto visual del primer botón (el botón "play") en la capa buttons. Por ejemplo, puede dibujar un óvalo horizontal con texto encima.
2. Con la herramienta Selección, seleccione todos los elementos gráficos del botón individual.
3. En el menú principal, elija Modificar > Convertir en símbolo.
4. En el cuadro de diálogo, elija Botón como tipo de símbolo, asigne un nombre al símbolo y haga clic en Aceptar.
5. Con el botón seleccionado, asigne al botón el nombre de instancia **playButton** en el inspector de propiedades.
6. Repita los pasos 1 a 5 para crear el botón que llevará al usuario a la página principal del autor. Asigne a este botón el nombre **homeButton**.

Escritura del código

El código ActionScript para esta aplicación puede dividirse en tres grupos de funcionalidad, aunque se escribirá todo en el mismo lugar. Las tres tareas que el código debe realizar son:

- Detener la cabeza lectora en cuanto se cargue el archivo SWF (cuando la cabeza lectora llegue al Fotograma 1).
- Detectar un evento para iniciar la reproducción del archivo SWF cuando el usuario haga clic en el botón play.
- Detectar un evento para enviar el navegador al URL apropiado cuando el usuario haga clic en el botón vinculado a la página de inicio del autor.

Para crear el código necesario para detener la cabeza lectora cuando llegue al Fotograma 1:

1. Seleccione el fotograma clave en el Fotograma 1 de la capa actions.
2. Para abrir el panel Acciones, en el menú principal, elija Ventana > Acciones.
3. En el panel Script, escriba el código siguiente:

```
stop();
```

Para escribir código para iniciar la animación cuando se haga clic en el botón play:

1. Al final del código escrito en los pasos anteriores, añada dos líneas vacías.
2. Escriba el código siguiente al final del script:

```
function startMovie(event:MouseEvent):void
{
    this.play();
}
```

   Este código define una función denominada `startMovie()`. Cuando se llama a `startMovie()`, hace que se inicie la reproducción de la línea de tiempo principal.

3. En la línea que sigue al código añadido en el paso anterior, escriba esta línea de código:

```
playButton.addEventListener(MouseEvent.CLICK, startMovie);
```

   Esta línea de código registra la función `startMovie()` como un detector del evento `click` de `playButton`. Es decir, hace que siempre que se haga clic en el botón `playButton`, se llame a la función `startMovie()`.

Para escribir código que envíe el navegador a una dirección URL cuando se haga clic en el botón vinculado a la página principal:

1 Al final del código escrito en los pasos anteriores, añada dos líneas vacías.

2 Escriba el código siguiente al final del script:

```
function gotoAuthorPage(event:MouseEvent):void
{
    var targetURL:URLRequest = new URLRequest("http://example.com/");
    navigateToURL(targetURL);
}
```

Este código define una función denominada `gotoAuthorPage()`. Esta función crea primero una instancia de URLRequest que representa el URL http://example.com/ y, a continuación, pasa el URL a la función `navigateToURL()`, lo que hace que el navegador del usuario abra dicho URL.

3 En la línea que sigue al código añadido en el paso anterior, escriba esta línea de código:

```
homeButton.addEventListener(MouseEvent.CLICK, gotoAuthorPage);
```

Esta línea de código registra la función `gotoAuthorPage()` como un detector del evento `click` de `homeButton`. Es decir, hace que siempre que se haga clic en el botón `homeButton`, se llame a la función `gotoAuthorPage()`.

Probar la aplicación

Al llegar a este punto, la aplicación ya debe ser completamente funcional. Pruébela.

Para probar la aplicación:

1 En el menú principal, elija Control > Probar película. Flash crea el archivo SWF y lo abre en una ventana de Flash Player.

2 Pruebe ambos botones para asegurarse de que funcionan correctamente.

3 Si los botones no funcionan, puede comprobar lo siguiente:

- ¿Tienen los botones nombres de instancia distintos?
- ¿Las llamadas al método `addEventListener()` utilizan los mismos nombres que los nombres de instancia de los botones?
- ¿Se utilizan los nombres de evento correctos en las llamadas al método `addEventListener()`?
- ¿Se ha especificado el parámetro correcto para cada una de las funciones? (Ambos deben tener un solo parámetro con el tipo de datos MouseEvent.)

Estos errores, y otros errores posibles, deben producir un mensaje de error cuando se elija el comando Probar película o cuando se haga clic en el botón. Vea si hay errores de compilador en el panel Errores del compilador (los que se producen al elegir Probar película) y si hay errores de tiempo de ejecución en el panel Salida (errores que se producen durante la reproducción del archivo SWF, como al hacer clic en un botón).

Creación de aplicaciones con ActionScript

El proceso de escritura de ActionScript para crear una aplicación implica algo más que el simple conocimiento de la sintaxis y los nombres de las clases que se van a utilizar. Si bien la mayor parte de la información de este manual está enfocada hacia esos dos temas (la sintaxis y la utilización de las clases de ActionScript), también es importante saber qué programas se pueden usar para escribir ActionScript, cómo puede organizarse e incluirse el código ActionScript en una aplicación y qué pasos hay que seguir para desarrollar una aplicación ActionScript.

Opciones para organizar el código

Se puede utilizar código ActionScript 3.0 para crear desde sencillas animaciones gráficas hasta complejos sistemas de procesamiento de transacciones cliente-servidor. Dependiendo del tipo de aplicación que se cree, se preferirá usar una o varias de las siguientes formas posibles de incluir código ActionScript en un proyecto.

Almacenamiento de código en fotogramas de una línea de tiempo de Flash

En el entorno de edición de Flash, es posible añadir código ActionScript a cualquier fotograma de una línea de tiempo. Este código se ejecutará mientras se reproduce la película, cuando la cabeza lectora alcance dicho fotograma.

La colocación del código ActionScript en fotogramas es una forma sencilla de añadir comportamientos a las aplicaciones incorporadas en la herramienta de edición de Flash. Se puede añadir código a cualquier fotograma de la línea de tiempo principal o a cualquier fotograma de la línea de tiempo de cualquier símbolo MovieClip. No obstante, esta flexibilidad tiene un coste. Cuando se crean aplicaciones de mayor tamaño, es fácil perder el rastro de los scripts contenidos en cada fotograma. Con el tiempo, esto puede dificultar el mantenimiento de la aplicación.

Muchos desarrolladores, para simplificar la organización de su código ActionScript en el entorno de edición de Flash, incluyen código únicamente en el primer fotograma de una línea de tiempo o en una capa específica del documento de Flash. De esta forma resulta más sencillo localizar y mantener el código en los archivos FLA de Flash. Sin embargo, para utilizar el mismo código en otro proyecto de Flash, es necesario copiar y pegar el código en el nuevo archivo.

Si desea poder reutilizar el código ActionScript en futuros proyectos de Flash, es recomendable almacenar el código en archivos de ActionScript externos (archivos de texto con la extensión .as).

Almacenamiento de código en archivos de ActionScript

Si el proyecto contiene una cantidad importante de código ActionScript, la mejor forma de organizar el código es en archivos de código fuente ActionScript independientes (archivos de texto con la extensión .as). Un archivo de ActionScript puede estructurarse de una o dos formas, dependiendo del uso que se le quiera dar en la aplicación.

- Código ActionScript no estructurado: líneas de código ActionScript, incluidas sentencias o definiciones de funciones, escritas como si se introdujeran directamente en un script de la línea de tiempo, un archivo MXML, etc.

  Para acceder al código ActionScript escrito de este modo, es preciso utilizar la sentencia `include` en ActionScript o la etiqueta `<mx:Script>` en Adobe MXML de Flex. La sentencia `include` de ActionScript hace que el contenido de un archivo de ActionScript externo se inserte en una ubicación específica y en un ámbito determinado de un script, como si se introdujera allí directamente. En el lenguaje MXML de Flex, la etiqueta `<mx:Script>` permite especificar un atributo de origen que identifica un archivo de ActionScript externo que se cargará en ese punto de la aplicación. Por ejemplo, la siguiente etiqueta cargará un archivo de ActionScript externo denominado Box.as:

  ```
  <mx:Script source="Box.as" />
  ```

- Definición de clase de ActionScript: definición de una clase de ActionScript, incluidas sus definiciones de métodos y propiedades.

  Cuando se define una clase, para obtener acceso al código ActionScript de la clase, se puede crear una instancia de la clase y utilizar sus propiedades, métodos y eventos, tal y como se haría con cualquiera de las clases de ActionScript incorporadas. Esto se realiza en dos partes:

  - Utilizar la sentencia `import` para especificar el nombre completo de la clase, de modo que el compilador de ActionScript sepa dónde encontrarlo. Por ejemplo, si se desea utilizar la clase MovieClip en ActionScript, primero se debe importar esa clase con su nombre completo, incluido el paquete y la clase:

    ```
    import flash.display.MovieClip;
    ```

    Como alternativa, se puede importar el paquete que contiene la clase MovieClip, que equivale a escribir sentencias `import` independientes para cada clase del paquete:

```
import flash.display.*;
```

Las únicas excepciones a la regla que dicta que una clase debe importarse si se hace referencia a ella en el código son las clases de nivel superior, que no se definen en un paquete.

Nota: *en Flash, las clases incorporadas (en los paquetes flash.*) se importan automáticamente en los scripts adjuntos a fotogramas de la línea de tiempo. Sin embargo, cuando se escriben clases propias o si se trabaja con los componentes de edición de Flash (los paquetes fl.*) o se trabaja en Flex, será necesario importar de forma explícita cualquier clase para poder escribir código que cree instancias de dicha clase.*

- Escribir código referido específicamente al nombre de clase (normalmente declarando una variable con esa clase como tipo de datos y creando una instancia de la clase para almacenarla en la variable). Al hacer referencia a otro nombre de clase en el código ActionScript, se indica al compilador que cargue la definición de dicha clase. Si se toma como ejemplo una clase externa denominada Box, esta sentencia provoca la creación de una nueva instancia de la clase Box:

  ```
  var smallBox:Box = new Box(10,20);
  ```

  Cuando el compilador se encuentra con la referencia a la clase Box por primera vez, busca el código fuente cargado para localizar la definición de la clase Box.

Selección de la herramienta adecuada

En función de las necesidades del proyecto y de los recursos disponibles, se puede utilizar una de las herramientas (o varias herramientas conjuntas) para escribir y editar el código ActionScript.

Herramienta de edición de Flash

Además de las capacidades de creación de animación y gráficos, Adobe Flash CS4 Professional incluye herramientas para trabajar con código ActionScript, ya sea asociado a los elementos de un archivo FLA como a archivos externos que sólo contienen código ActionScript. La herramienta de edición de Flash es ideal para los proyectos que contienen una cantidad importante de animación o vídeo, o los proyectos donde el usuario desea crear la mayoría de los activos gráficos, concretamente los proyectos con una mínima interacción del usuario o funcionalidad que requiera ActionScript. Otro motivo para elegir la herramienta de edición de Flash para desarrollar los proyectos de ActionScript es que se prefiera crear activos visuales y escribir código en la misma aplicación. Quizá también se prefiera usar la herramienta de edición de Flash si se van a utilizar componentes de interfaz de usuario creados previamente, pero las prioridades básicas del proyecto sean reducir el tamaño de los archivos SWF o facilitar la aplicación de aspectos visuales.

Adobe Flash CS4 Professional incluye dos herramientas para escribir código ActionScript:

- Panel Acciones: este panel, disponible cuando se trabaja en un archivo FLA, permite escribir código ActionScript asociado a los fotogramas de una línea de tiempo.
- Ventana Script: la ventana Script es un editor de texto dedicado para trabajar con archivos de código ActionScript (.as).

Flex Builder

Adobe Flex Builder es la principal herramienta para crear proyectos con la arquitectura Flex. Además de sus herramientas de edición de MXML y diseño visual, Flex Builder incluye un editor de ActionScript completo, de modo que puede usarse para crear proyectos de Flex o sólo de ActionScript. Las aplicaciones Flex presentan varias ventajas, como un amplio conjunto de controles de interfaz de usuario predefinidos, controles de diseño dinámicos y flexibles, y mecanismos incorporados para trabajar con orígenes de datos externos y vincular datos externos con elementos de interfaz de usuario. Sin embargo, debido al código adicional necesario para proporcionar estas funciones, las aplicaciones Flex pueden tener un tamaño de archivo SWF más grande y presentan una mayor dificultad que sus equivalentes de Flash a la hora de cambiar completamente los aspectos aplicados.

Se recomienda usar Flex Builder para crear con Flex complejas y completas aplicaciones para Internet basadas en datos, y para editar código ActionScript, editar código MXML y diseñar visualmente la aplicación con una sola herramienta.

Editor de ActionScript de terceros

Dado que los archivos ActionScript (.as) se almacenan como archivos de texto sencillo, cualquier programa que sea capaz de editar archivos de texto simple se puede usar para escribir archivos ActionScript. Además de los productos ActionScript de Adobe, se han creado varios programas de edición de texto de terceros con funciones específicas de ActionScript. Se pueden escribir archivos MXML o clases de ActionScript con cualquier programa editor de texto. A partir de estos archivos, se puede crear una aplicación SWF (ya sea una aplicación Flex o sólo de ActionScript) con la ayuda del SDK de Flex, que incluye las clases de la arquitectura Flex además del compilador Flex. Como alternativa, muchos desarrolladores utilizan un editor de ActionScript de terceros para escribir las clases de ActionScript y la herramienta de edición de Flash para crear el contenido gráfico.

El usuario puede optar por utilizar un editor de ActionScript de terceros si:

- Prefiere escribir código ActionScript en un programa independiente y diseñar los elementos visuales en Flash.
- Utiliza una aplicación para programar con un lenguaje distinto de ActionScript (por ejemplo, para crear páginas HTML o aplicaciones en otro lenguaje de programación) y desea usar la misma aplicación para el código ActionScript.
- Desea crear proyectos de Flex o sólo de ActionScript con el SDK de Flex, sin tener que utilizar Flash o Flex Builder.

Entre los editores de código que proporcionan funciones específicas de ActionScript, cabe destacar:

- Adobe Dreamweaver® CS4
- ASDT
- FDT
- FlashDevelop
- PrimalScript
- SE|PY

Proceso de desarrollo de ActionScript

Independientemente del tamaño del proyecto de ActionScript, la utilización de un proceso para diseñar y desarrollar la aplicación permitirá trabajar con mayor eficacia. En los siguientes pasos se describe un proceso de desarrollo básico para crear una aplicación con ActionScript 3.0:

1 Diseñe la aplicación.

Debe describir la aplicación de alguna forma antes de empezar a crearla.

2 Escriba el código ActionScript 3.0.

Puede crear código ActionScript con Flash, Flex Builder, Dreamweaver o un editor de texto.

3 Cree un archivo de aplicación Flash o Flex para ejecutar el código.

En la herramienta de edición de Flash, esto implica crear un nuevo archivo FLA, establecer la configuración de publicación, añadir componentes de interfaz de usuario a la aplicación y hacer referencia al código ActionScript. En el entorno de desarrollo de Flex, la creación de un nuevo archivo de aplicación implica definir la aplicación, añadir componentes de interfaz de usuario con MXML y hacer referencia al código ActionScript.

4 Publique y pruebe la aplicación ActionScript.

Esto implica ejecutar la aplicación desde la herramienta de edición de Flash o el entorno de desarrollo de Flex, y comprobar que realiza todo lo previsto.

Debe tenerse en cuenta que estos pasos no tienen por qué seguir este orden necesariamente y que tampoco es necesario finalizar completamente uno de los pasos antes de poder trabajar en otro. Por ejemplo, se puede diseñar una pantalla de la aplicación (paso 1) y luego crear los gráficos, botones y otros elementos (paso 3) antes de escribir el código ActionScript (paso 2) y probarlo (paso 4). O bien, se puede realizar parte del diseño y luego añadir un botón o elemento de interfaz en un momento dado, escribir código ActionScript para cada uno de estos elementos y probarlos. Aunque resulta útil recordar estas cuatro fases del proceso de desarrollo, en una situación real suele ser más eficaz ir pasando de una fase a otra según convenga.

Creación de clases personalizadas

El proceso de creación de clases para usarlas en los proyectos puede parecer desalentador. Sin embargo, la tarea más difícil de la creación de una clase es su diseño, es decir, la identificación de los métodos, propiedades y eventos que va a incluir.

Estrategias de diseño de una clase

El tema del diseño orientado a objetos es complejo; algunas personas han dedicado toda su carrera al estudio académico y la práctica profesional de esta disciplina. Sin embargo, a continuación se sugieren algunos enfoques que pueden ayudarle a comenzar.

1 Piense en la función que desempeñarán las instancias de esta clase en la aplicación. Normalmente, los objetos desempeñan una de estas tres funciones:

- Objeto de valor: estos objetos actúan principalmente como contenedores de datos, es decir, que es probable que tengan varias propiedades y pocos métodos (o algunas veces ninguno). Normalmente son representaciones de código de elementos claramente definidos, como una clase Song (que representa una sola canción real) o una clase Playlist (que representa un grupo conceptual de canciones) en una aplicación de reproductor de música.
- Objetos de visualización: son objetos que aparecen realmente en la pantalla. Por ejemplo, elementos de interfaz de usuario como una lista desplegable o una lectura de estado, o elementos gráficos como las criaturas de un videojuego, etc.
- Estructura de aplicación: estos objetos desempeñan una amplia gama de funciones auxiliares en la lógica o el procesamiento llevado a cabo por las aplicaciones. Algunos ejemplos son: un objeto que realice determinados cálculos en una simulación biológica, un objeto responsable de sincronizar valores entre un control de dial y una lectura de volumen en una aplicación de reproductor de música, uno que administre las reglas de un videojuego u otro que cargue una imagen guardada en una aplicación de dibujo.

2 Decida la funcionalidad específica que necesitará la clase. Los distintos tipos de funcionalidad suelen convertirse en los métodos de la clase.

3 Si se prevé que la clase actúe como un objeto de valor, decida los datos que incluirán las instancias. Estos elementos son buenos candidatos para las propiedades.

4 Dado que la clase se diseña específicamente para el proyecto, lo más importante es proporcionar la funcionalidad que necesita la aplicación. Quizá le sirva de ayuda formularse estas preguntas:

- ¿Qué elementos de información almacenará, rastreará y manipulará la aplicación? Esta decisión le ayudará a identificar los posibles objetos de valor y las propiedades.
- ¿Qué conjuntos de acciones deberán realizarse, por ejemplo, al cargar la aplicación por primera vez, al hacer clic en un botón concreto o al detener la reproducción de una película? Estos serán buenos candidatos para los métodos (o propiedades, si las "acciones" implican simplemente cambiar valores individuales).
- En cualquier acción determinada, ¿qué información necesitará saber la clase para realizar dicha acción? Estos elementos de información se convierten en los parámetros del método.
- Mientras la aplicación lleva a cabo su trabajo, ¿qué cambiará en la clase que deba ser conocido por las demás partes de la aplicación? Éstos son buenos candidatos para los eventos.

5 Si ya existe un objeto similar al que necesita, a no ser que carezca de alguna funcionalidad adicional que desee añadir, considere la posibilidad de crear una subclase (una clase que se basa en la funcionalidad de una clase existente, en lugar de definir toda su propia funcionalidad). Por ejemplo, si desea crear una clase que sea un objeto visual en la pantalla, deseará usar el comportamiento de uno de los objetos de visualización existentes (por ejemplo, Sprite o MovieClip) como base de la clase. En ese caso, MovieClip (o Sprite) sería la *clase base* y su clase sería una ampliación de dicha clase. Para obtener más información sobre cómo crear una subclase, consulte "Herencia" en la página 111.

Escritura del código de una clase

Cuando ya tenga un plan de diseño para la clase, o al menos alguna idea de la información de la que deberá hacer un seguimiento y de las acciones que necesitará realizar, la sintaxis real de escritura de una clase es bastante directa.

A continuación se describen los pasos mínimos para crear su propia clase de ActionScript:

1 Abra un nuevo documento de texto en un programa específico de ActionScript como Flex Builder o Flash, en una herramienta de programación general como Dreamweaver o en cualquier programa que permita trabajar con documentos de texto simple.

2 Introduzca una sentencia `class` para definir el nombre de la clase. Para ello, introduzca las palabras `public class` y, a continuación, el nombre de la clase, seguido de llaves de apertura y cierre que rodearán el contenido de la clase (las definiciones de métodos y propiedades). Por ejemplo:

```
public class MyClass
{
}
```

La palabra `public` indica que es posible acceder a la clase desde cualquier otro código. Para conocer otras alternativas, consulte "Atributos del espacio de nombres de control de acceso" en la página 97.

3 Escriba una sentencia `package` para indicar el nombre del paquete en el que se encontrará la clase. La sintaxis es la palabra `package`, seguida del nombre completo del paquete, seguido de llaves de apertura y cierre (que rodearán el bloque de sentencias `class`). Por ejemplo, se cambiaría el código del paso anterior por el siguiente:

```
package mypackage
{
    public class MyClass
    {
    }
}
```

4 Defina cada propiedad de la clase con la sentencia `var` en el cuerpo de la clase; la sintaxis es la misma que se usa para declarar cualquier variable (añadiendo el modificador `public`). Por ejemplo, si se añaden estas líneas entre los paréntesis de apertura y cierre de la definición de clase, se crearán las propiedades `textVariable`, `numericVariable` y `dateVariable`:

```
public var textVariable:String = "some default value";
public var numericVariable:Number = 17;
public var dateVariable:Date;
```

5 Defina cada método de la clase con la misma sintaxis empleada para definir una función. Por ejemplo:

- Para crear un método `myMethod()`, introduzca:

```
public function myMethod(param1:String, param2:Number):void
{
    // do something with parameters
}
```

- Para crear un constructor (el método especial al que se llama como parte del proceso de creación de una instancia de una clase), cree un método cuyo nombre coincida exactamente con el nombre de la clase:

```
public function MyClass()
{
    // do stuff to set initial values for properties
    // and otherwise set up the object
    textVariable = "Hello there!";
    dateVariable = new Date(2001, 5, 11);
}
```

  Si no incluye un método constructor en la clase, el compilador creará automáticamente un constructor vacío (sin parámetros ni sentencias) en la clase.

Hay algunos otros elementos de la clase que pueden definirse y que son más complicados.

- Los *descriptores de acceso* son un cruce especial entre un método y una propiedad. Cuando se escribe el código para definir la clase, el descriptor de acceso se escribe como un método, de forma que es posible realizar varias acciones, en lugar de simplemente leer o asignar un valor, que es todo lo que puede hacerse cuando se define una propiedad. Sin embargo, cuando se crea una instancia de la clase, se trata al descriptor de acceso como una propiedad y sólo se utiliza el nombre para leer o asignar el valor. Para obtener más información, consulte "Métodos descriptores de acceso (captador y definidor)" en la página 104.
- En ActionScript, los eventos no se definen con una sintaxis específica. En lugar de eso, los eventos de la clase se definen con la funcionalidad de la clase EventDispatcher para realizar un seguimiento de los detectores de eventos y notificarles los eventos. Para obtener más información sobre la creación de eventos en sus propias clases, consulte "Gestión de eventos" en la página 254.

Ejemplo: Creación de una aplicación básica

Es posible crear archivos de código fuente ActionScript externos con una extensión .as utilizando Flash, Flex Builder, Dreamweaver o cualquier editor de texto.

ActionScript 3.0 puede utilizarse en varios entornos de desarrollo de aplicaciones, como las herramientas de edición de Flash y Flex Builder.

En esta sección se indican los pasos necesarios para crear y mejorar una sencilla aplicación ActionScript 3.0 con la herramienta de edición de Flash o con Flex Builder. La aplicación que se va a crear presenta un patrón sencillo de utilización de archivos de clases de ActionScript 3.0 externos en aplicaciones Flash y Flex. Dicho patrón se utilizará en todas las demás aplicaciones de ejemplo de este manual.

Diseño de una aplicación ActionScript

Debería tener alguna idea de la aplicación que desea crear antes de empezar a crearla.

La representación del diseño puede ser tan sencilla como el nombre de la aplicación y una breve declaración del propósito de la misma o tan complicada como un conjunto de documentos de requisitos con numerosos diagramas de Lenguaje de modelado unificado (UML). Aunque este manual no trata con detalle el tema del diseño de software, es importante recordar que el diseño de la aplicación es un paso fundamental del desarrollo de las aplicaciones ActionScript.

El primer ejemplo de una aplicación ActionScript es una aplicación "Hello World" estándar, de modo que su diseño es muy sencillo:

- La aplicación se denominará HelloWorld.
- Mostrará un solo campo de texto con las palabras "Hello World!".
- Para poder reutilizarla fácilmente, utilizará una sola clase orientada a objetos, denominada Greeter, que puede usarse desde un documento de Flash o una aplicación Flex.
- Después de crear una versión básica de la aplicación, deberá añadir funcionalidad para que haga que el usuario introduzca un nombre de usuario y que la aplicación compruebe el nombre en una lista de usuarios conocidos.

Teniendo en cuenta esta definición concisa, ya puede empezar a crear la aplicación.

Creación del proyecto HelloWorld y de la clase Greeter

Según el propósito del diseño de la aplicación Hello World, el código debería poder reutilizarse fácilmente. Teniendo esto en cuenta, la aplicación utiliza una sola clase orientada a objetos, denominada Greeter, que se usa desde una aplicación creada en Flex Builder o la herramienta de edición de Flash.

Para crear la clase Greeter en la herramienta de edición de Flash:

1. En la herramienta de edición de Flash, seleccione Archivo > Nuevo.
2. En el cuadro de diálogo Nuevo documento, seleccione Archivo ActionScript y haga clic en Aceptar.

   Aparecerá una nueva ventana de edición de ActionScript.
3. Seleccione Archivo > Guardar. Seleccione la carpeta en la que desea almacenar la aplicación, asigne el nombre **Greeter.as** al archivo ActionScript y haga clic en Aceptar.

   Para continuar, consulte la sección “Añadir código a la clase Greeter” en la página 31.

Añadir código a la clase Greeter

La clase Greeter define un objeto, `Greeter`, que podrá utilizar en la aplicación HelloWorld.

Para añadir código a la clase Greeter:

1. Introduzca el código siguiente en el nuevo archivo:

```
package
{
    public class Greeter
    {
        public function sayHello():String
        {
            var greeting:String;
            greeting = "Hello World!";
            return greeting;
        }
    }
}
```

La clase Greeter incluye un único método `sayHello()`, el cual devuelve una cadena con el texto "Hello World!".

2 Seleccione Archivo > Guardar para guardar este archivo de ActionScript.

Ya se puede utilizar la clase Greeter en la aplicación.

Creación de una aplicación que utilice el código ActionScript

La clase Greeter que ha creado define un conjunto de funciones de software con contenido propio, pero no representa una aplicación completa. Para utilizar la clase, necesita crear un documento de Flash o una aplicación Flex.

La aplicación HelloWorld crea una nueva instancia de la clase Greeter. A continuación se explica cómo asociar la clase Greeter a la aplicación.

Para crear una aplicación ActionScript mediante la herramienta de edición de Flash:

1 Seleccione Archivo > Nuevo.

2 En el cuadro de diálogo Nuevo documento, seleccione Documento de Flash y haga clic en Aceptar.

Aparece una nueva ventana de Flash.

3 Seleccione Archivo > Guardar. Seleccione la misma carpeta que contiene el archivo de clase Greeter.as, asigne al documento de Flash el nombre **HelloWorld.fla** y haga clic en Aceptar.

4 En la paleta Herramientas de Flash, seleccione la herramienta Texto y arrastre el cursor por el escenario para definir un nuevo campo de texto, con una anchura de aproximadamente 300 píxeles y una altura de unos 100 píxeles.

5 En el panel Propiedades, con el campo de texto todavía seleccionado en el escenario, establezca "Texto dinámico" como tipo de texto y escriba **mainText** como nombre de instancia del campo de texto.

6 Haga clic en el primer fotograma de la línea de tiempo principal.

7 En el panel Acciones, escriba el siguiente script:

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello();
```

8 Guarde el archivo.

Para continuar, consulte la sección “Publicación y prueba de la aplicación ActionScript” en la página 32.

Publicación y prueba de la aplicación ActionScript

El desarrollo de software es un proceso repetitivo. Se escribe código, se intenta compilar y se edita hasta que se compile sin problemas. Se ejecuta la aplicación compilada, se prueba para ver si cumple con el diseño previsto y, si no es así, se edita de nuevo el código hasta que lo cumple. Los entornos de desarrollo de Flash y Flex Builder ofrecen diversas formas de publicar, probar y depurar las aplicaciones.

A continuación se explican los pasos básicos para probar la aplicación HelloWorld en cada entorno.

Para publicar y probar una aplicación ActionScript mediante la herramienta de edición de Flash:

1 Publique la aplicación y vea si aparecen errores de compilación. En la herramienta de edición de Flash, seleccione Control > Probar película para compilar el código ActionScript y ejecutar la aplicación HelloWorld.

2 Si aparecen errores o advertencias en la ventana Salida al probar la aplicación, corrija las causas de estos errores en el archivo HelloWorld.fla o HelloWorld.as y pruebe de nuevo la aplicación.

3 Si no se producen errores de compilación, verá una ventana de Flash Player en la que se mostrará la aplicación Hello World.

Acaba de crear una aplicación orientada a objetos sencilla, pero completa, que utiliza ActionScript 3.0. Para continuar, consulte la sección "Mejora de la aplicación HelloWorld" en la página 33.

Mejora de la aplicación HelloWorld

Para hacer la aplicación un poco más interesante, hará que pida y valide un nombre de usuario en una lista predefinida de nombres.

En primer lugar, deberá actualizar la clase Greeter para añadir funcionalidad nueva. A continuación, actualizará la aplicación para utilizar la nueva funcionalidad.

Para actualizar el archivo Greeter.as:

1 Abra el archivo Greeter.as.

2 Cambie el contenido del archivo por lo siguiente (las líneas nuevas y cambiadas se muestran en negrita):

```
package
{
    public class Greeter
    {
        /**
         * Defines the names that should receive a proper greeting.
         */
        public static var validNames:Array = ["Sammy", "Frank", "Dean"];

        /**
         * Builds a greeting string using the given name.
         */
        public function sayHello(userName:String = ""):String
        {
            var greeting:String;
            if (userName == "")
            {
                greeting = "Hello. Please type your user name, and then press
                            the Enter key.";
            }
            else if (validName(userName))
            {
                greeting = "Hello, " + userName + ".";
            }
            else
            {
                greeting = "Sorry " + userName + ", you are not on the list.";
```

```
            }
            return greeting;
        }

        /**
         * Checks whether a name is in the validNames list.
         */
        public static function validName(inputName:String = ""):Boolean
        {
            if (validNames.indexOf(inputName) > -1)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

La clase Greeter tiene ahora varias funciones nuevas:

- El conjunto `validNames` enumera los nombres de usuario válidos. El conjunto se inicializa como una lista de tres nombres cuando se carga la clase Greeter.
- El método `sayHello()` acepta ahora un nombre de usuario y cambia el saludo en función de algunas condiciones. Si `userName` es una cadena vacía (`""`), la propiedad `greeting` se define de forma que pida un nombre al usuario. Si el nombre de usuario es válido, el saludo se convierte en `"Hello, `*userName*`"`. Finalmente, si no se cumple alguna de estas dos condiciones, la variable `greeting` se define como `"Sorry, `*userName*`, you are not on the list"`.
- El método `validName()` devuelve `true` si `inputName` se encuentra en el conjunto `validNames` y `false` si no se encuentra. La sentencia `validNames.indexOf(inputName)` comprueba cada una de las cadenas del conjunto `validNames` con respecto a la cadena `inputName`. El método `Array.indexOf()` devuelve la posición de índice de la primera instancia de un objeto en un conjunto o el valor -1 si el objeto no se encuentra en el conjunto.

A continuación editará el archivo de Flash o Flex que hace referencia a esta clase de ActionScript.

Para modificar la aplicación ActionScript mediante la herramienta de edición de Flash:

1. Abra el archivo HelloWorld.fla.
2. Modifique el script en el Fotograma 1 de forma que se pase una cadena vacía (`""`) al método `sayHello()` de la clase Greeter:

   ```
   var myGreeter:Greeter = new Greeter();
   mainText.text = myGreeter.sayHello("");
   ```

3. Seleccione la herramienta Texto en la paleta Herramientas y cree dos nuevos campos de texto en el escenario, uno junto al otro, y debajo del campo de texto `mainText` existente.
4. En el primer campo de texto nuevo, escriba **User Name:** como etiqueta.
5. Seleccione el otro campo de texto nuevo y, en el inspector de propiedades, seleccione InputText como tipo del campo de texto. Seleccione Línea única como tipo de línea. Escriba **textIn** como nombre de instancia.
6. Haga clic en el primer fotograma de la línea de tiempo principal.
7. En el panel Acciones, añada las líneas siguientes al final del script existente:

```
mainText.border = true;
textIn.border = true;

textIn.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);

function keyPressed(event:KeyboardEvent):void
{
    if (event.keyCode == Keyboard.ENTER)
    {
        mainText.text = myGreeter.sayHello(textIn.text);
    }
}
```

El código nuevo añade la siguiente funcionalidad:

- Las dos primeras líneas sólo definen los bordes de dos campos de texto.
- Un campo de texto de entrada, como el campo `textIn`, tiene un conjunto de eventos que puede distribuir. El método `addEventListener()` permite definir una función que se ejecuta cuando se produce un tipo de evento. En este caso, el evento es presionar una tecla del teclado.
- La función personalizada `keyPressed()` comprueba si la tecla presionada es la tecla Intro. De ser así, llama al método `sayHello()` del objeto `myGreeter` y le pasa el texto del campo de texto `textIn` como parámetro. El método devuelve una cadena de saludo basada en el valor pasado. Después se asigna la cadena devuelta a la propiedad `text` del campo de texto `mainText`.

A continuación se muestra el script completo para el Fotograma 1:

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");

mainText.border = true;
textIn.border = true;

textIn.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);

function keyPressed(event:KeyboardEvent):void
{
    if (event.keyCode == Keyboard.ENTER)
    {
        mainText.text = myGreeter.sayHello(textIn.text);
    }
}
```

8 Guarde el archivo.

9 Seleccione Control > Probar película para ejecutar la aplicación.

Cuando ejecute la aplicación se le pedirá que escriba un nombre de usuario. Si es válido (Sammy, Frank o Dean), la aplicación mostrará el mensaje de confirmación "hello".

Ejecución de ejemplos posteriores

Después de desarrollar y ejecutar la aplicación "Hello World" de ActionScript 3.0, debería haber adquirido los conocimientos básicos que necesita para ejecutar los demás ejemplos de código presentados en este manual.

Prueba de los listados de código de ejemplo del capítulo

A medida que avance en el manual, deseará probar los listados de código de ejemplo con los que se ilustran los distintos temas. Para probarlos puede tener que mostrar el valor de las variables en determinados puntos del programa o bien ver el contenido mostrado en pantalla o interactuar con él. Para las pruebas de contenido visual o de interacción, los elementos necesarios se describirán antes (o dentro) del listado de código; sólo tendrá que crear un documento con los elementos descritos por orden para probar el código. En caso de que desee ver el valor de una variable en un punto determinado del programa, puede hacerlo de varias maneras distintas. Una manera es utilizar un depurador, como los integrados en Flex Builder y en Flash. Sin embargo, para las pruebas sencillas puede ser más fácil simplemente imprimir los valores de las variables en algún lugar en que pueda verlas.

Los pasos siguientes le ayudarán a crear un documento de Flash con el que podrá probar un listado de código y ver valores de las variables:

Para crear un documento de Flash a fin de probar los ejemplos del capítulo:

1. Cree un nuevo documento de Flash y guárdelo en el disco duro.
2. Para mostrar valores de prueba en un campo de texto en el escenario, active la herramienta Texto y cree un nuevo campo de texto dinámico en el escenario. Le resultará útil tener un campo de texto ancho y alto, con el tipo de línea establecido en Multilínea. En el inspector de propiedades, asigne al campo de texto un nombre de instancia (por ejemplo, "outputText"). Para poder escribir valores en el campo de texto hay que añadir al ejemplo código que llame al método `appendText()` (descrito a continuación).
3. Como alternativa, se puede añadir una llamada a función `trace()` al listado de código (como se indica más abajo) para ver los resultados del ejemplo.
4. Para probar un ejemplo determinado, copie el listado de código en el panel Acciones; si es necesario, añada una llamada a la función `trace()` o añada un valor al campo de texto utilizando su método `appendText()`.
5. En el menú principal, elija Control > Probar película para crear un archivo SWF y ver los resultados.

Dado que este enfoque se utiliza para ver valores de variables, hay dos formas de consultar fácilmente los valores de las variables conforme se prueban los ejemplos: escribiendo valores en una instancia de campo de texto en el escenario o utilizando la función `trace()` para imprimir valores en el panel Salida.

- La función `trace()` de ActionScript: la función `trace()` de ActionScript escribe los valores de los parámetros que recibe (ya sean variables o expresiones literales) en el panel Salida. Muchos de los listados de ejemplo de este manual ya incluyen una llamada a la función `trace()`, así que para esos listados sólo tendrá que copiar el código en el documento y probar el proyecto. Si desea utilizar `trace()` para probar el valor de una variable en un listado de código que no la incluye, sólo tiene que añadir una llamada a `trace()` al listado de código y pasarle la variable como parámetro. Por ejemplo, si encuentra un listado de código como el de este capítulo,

  ```
  var albumName:String = "Three for the money";
  ```

  puede copiar el código en el panel Acciones y añadir una llamada a la función `trace()` como ésta para probar el resultado del listado de código:

  ```
  var albumName:String = "Three for the money";
  trace("albumName =", albumName);
  ```

  Si ejecuta el programa, se imprimirá esta línea:

  ```
  albumName = Three for the money
  ```

  Cada llamada a la función `trace()` puede incluir varios parámetros, que se encadenan como una sola línea impresa. Al final de cada llamada a la función `trace()` se añade un salto de línea, por lo que distintas llamadas a la función `trace()` se imprimirán en líneas distintas.

- Un campo de texto en el escenario: si prefiere no utilizar la función `trace()`, puede añadir un campo de texto dinámico al escenario mediante la herramienta Texto y escribir valores en dicho campo para ver los resultados de un listado de código. Se puede utilizar el método `appendText()` de la clase TextField para añadir un valor String al final del contenido del campo de texto. Para acceder al campo de texto mediante código ActionScript, debe asignarle un nombre de instancia en el inspector de propiedades. Por ejemplo, si el campo de texto tiene el nombre de instancia `outputText`, se puede utilizar el código siguiente para comprobar el valor de la variable `albumName`:

  ```
  var albumName:String = "Three for the money";
  outputText.appendText("albumName = ");
  outputText.appendText(albumName);
  ```

  Este código escribirá el texto siguiente en el campo de texto denominado `outputText`:

  ```
  albumName = Three for the money
  ```

  Como muestra el ejemplo, el método `appendText()` añadirá el texto a la misma línea que el contenido anterior, por lo que se pueden añadir varias líneas a la misma línea de texto mediante varias llamadas a `appendText()`. Para forzar que el texto pase a la siguiente línea, puede añadir un carácter de nueva línea (`"\n"`):

  ```
  outputText.appendText("\n"); // adds a line break to the text field
  ```

  A diferencia de la función `trace()`, el método `appendText()` sólo acepta un valor como parámetro. Dicho valor debe ser una cadena (una instancia de String o un literal de cadena). Para imprimir el valor de una variable que no sea de tipo cadena, primero debe convertir el valor en una cadena. La forma más sencilla de hacer esto es llamar al método `toString()` del objeto:

  ```
  var albumYear:int = 1999;
  outputText.appendText("albumYear = ");
  outputText.appendText(albumYear.toString());
  ```

Ejemplos de final de capítulo

Al igual que este capítulo, la mayoría de los capítulos incluyen un ejemplo ilustrativo de fin de capítulo que relaciona muchos de los conceptos descritos. Sin embargo, a diferencia del ejemplo Hello World de este capítulo, los ejemplos no se presentarán en un formato de tutorial paso a paso. Se resaltará y explicará el código ActionScript 3.0 relevante de cada ejemplo, pero no se ofrecerán instrucciones para ejecutar los ejemplos en entornos de desarrollo específicos. Sin embargo, los archivos de ejemplo distribuidos con este manual incluirán todos los archivos necesarios para compilar fácilmente los ejemplos en el entorno de desarrollo elegido.

Capítulo 4: El lenguaje ActionScript y su sintaxis

ActionScript 3.0 consta del lenguaje ActionScript y la interfaz de programación de aplicaciones (API) de Adobe Flash Player. El lenguaje principal es la parte de ActionScript que define la sintaxis del lenguaje, así como los tipos de datos de nivel superior. ActionScript 3.0 proporciona acceso programado a Flash Player.

Este capítulo ofrece una breve introducción al lenguaje ActionScript y su sintaxis. Su lectura proporciona conocimientos básicos sobre cómo trabajar con tipos de datos y variables, utilizar la sintaxis correcta y controlar el flujo de datos de un programa.

Información general sobre el lenguaje

Los objetos constituyen la base del lenguaje ActionScript 3.0. Son sus componentes esenciales. Cada variable que se declare, cada función que se escriba y cada instancia de clase que se cree es un objeto. Se puede considerar que un programa ActionScript 3.0 es un grupo de objetos que realizan tareas, responden a eventos y se comunican entre sí.

Para los programadores que están familiarizados con la programación orientada a objetos (OOP) en Java o C++ los objetos son módulos que contienen dos tipos de miembros: datos almacenados en variables o propiedades miembro, y comportamiento al que se puede acceder a través de métodos. ActionScript 3.0 define los objetos de forma similar, aunque ligeramente distinta. En ActionScript 3.0, los objetos son simplemente colecciones de propiedades. Estas propiedades son contenedores que pueden contener no sólo datos, sino también funciones u otros objetos. Si se asocia una función a un objeto de esta manera, la función se denomina método.

Aunque la definición de ActionScript 3.0 puede parecer extraña a los programadores con experiencia en programación con Java o C++, en la práctica, la definición de tipos de objetos con clases de ActionScript 3.0 es muy similar a la manera de definir clases en Java o C++. La distinción entre las dos definiciones de objeto es importante al describir el modelo de objetos de ActionScript y otros temas avanzados, pero en la mayoría de las demás situaciones, el término *propiedades* se refiere a variables miembro de clase, no a métodos. La Referencia del lenguaje y componentes ActionScript 3.0 utiliza, por ejemplo, el término *propiedades* para hacer referencia a variables o propiedades de captores y definidores. El término *métodos* se utiliza para designar funciones que forman parte de una clase.

Una diferencia sutil entre las clases de ActionScript y las clases de Java o C++ es que, en ActionScript, las clases no son sólo entidades abstractas. Las clases de ActionScript se representan mediante *objetos de clase* que almacenan las propiedades y los métodos de la clase. Esto permite utilizar técnicas que pueden parecer extrañas a los programadores de Java y C++, como incluir sentencias o código ejecutable en el nivel superior de una clase o un paquete.

Otra diferencia entre las clases de ActionScript y las clases de Java o C++ es que cada clase de ActionScript tiene algo denominado *objetoprototipo*. En versiones anteriores de ActionScript, los objetos prototipo, vinculados entre sí en *cadenas de prototipos*, constituían en conjunto la base de toda la jerarquía de herencia de clases. Sin embargo, en ActionScript 3.0 los objetos prototipo desempeñan una función poco importante en el sistema de herencia. Pero el objeto prototipo puede ser útil como alternativa a las propiedades y los métodos estáticos si se desea compartir una propiedad y su valor entre todas las instancias de una clase.

En el pasado, los programadores expertos de ActionScript podían manipular directamente la cadena de prototipos con elementos especiales incorporados en el lenguaje. Ahora que el lenguaje proporciona una implementación más madura de una interfaz de programación basada en clases, muchos de estos elementos del lenguaje especiales, como `__proto__` y `__resolve`, ya no forman parte del lenguaje. Asimismo, las optimizaciones realizadas en el mecanismo de herencia interno, que aportan mejoras importantes de rendimiento en Flash Player y Adobe AIR, impiden el acceso directo al mecanismo de herencia.

Objetos y clases

En ActionScript 3.0, cada objeto se define mediante una clase. Una clase puede considerarse como una plantilla o un modelo para un tipo de objeto. Las definiciones de clase pueden incluir variables y constantes, que contienen valores de datos y métodos, que son funciones que encapsulan el comportamiento asociado a la clase. Los valores almacenados en propiedades pueden ser *valores simples* u otros objetos. Los valores simples son números, cadenas o valores booleanos.

ActionScript contiene diversas clases incorporadas que forman parte del núcleo del lenguaje. Algunas de estas clases incorporadas, como Number, Boolean y String, representan los valores simples disponibles en ActionScript. Otras, como las clases Array, Math y XML, definen objetos más complejos.

Todas las clases, tanto las incorporadas como las definidas por el usuario, se derivan de la clase Object. Para los programadores con experiencia previa en ActionScript, es importante tener en cuenta que el tipo de datos Object ya no es el tipo de datos predeterminado, aunque todas las demás clases se deriven de él. En ActionScript 2.0, las dos líneas de código siguientes eran equivalentes, ya que la ausencia de una anotación de tipo significaba que una variable era de tipo Object:

```
var someObj:Object;
var someObj;
```

En ActionScript 3.0 se introduce el concepto de variables sin tipo, que pueden designarse de las dos maneras siguientes:

```
var someObj:*;
var someObj;
```

Una variable sin tipo no es lo mismo que una variable de tipo Object. La principal diferencia es que las variables sin tipo pueden contener el valor especial `undefined`, mientras que una variable de tipo Object no puede contener ese valor.

Un programador puede definir sus propias clases mediante la palabra clave `class`. Las propiedades de clase se pueden definir de tres formas diferentes: las constantes se pueden definir con la palabra clave `const`, las variables con la palabra clave `var`, y las propiedades de captores y definidores con los atributos `get` y `set` de una declaración de método. Los métodos se declaran con la palabra clave `function`.

Para crear una instancia de una clase hay que utilizar el operador `new`. En el ejemplo siguiente se crea una instancia de la clase Date denominada `myBirthday`.

```
var myBirthday:Date = new Date();
```

Paquetes y espacios de nombres

Los conceptos de paquete y espacio de nombres están relacionados. Los paquetes permiten agrupar definiciones de clase de una manera que permite compartir código fácilmente y minimiza los conflictos de nomenclatura. Los espacios de nombres permiten controlar la visibilidad de los identificadores, como los nombres de propiedades y métodos, y pueden aplicarse a código tanto dentro como fuera de un paquete. Los paquetes permiten organizar los archivos de clase y los espacios de nombres permiten administrar la visibilidad de propiedades y métodos individuales.

Paquetes

En ActionScript 3.0, los paquetes se implementan con espacios de nombres, pero son un concepto distinto. Al declarar un paquete, se crea implícitamente un tipo especial de espacio de nombres que se conoce en tiempo de compilación. Si los espacios de nombres se crean explícitamente, no se conocerán necesariamente en tiempo de compilación.

En el ejemplo siguiente se utiliza la directiva `package` para crear un paquete sencillo que contiene una clase:

```
package samples
{
    public class SampleCode
    {
        public var sampleGreeting:String;
        public function sampleFunction()
        {
            trace(sampleGreeting + " from sampleFunction()");
        }
    }
}
```

El nombre de la clase de este ejemplo es SampleCode. Debido a que la clase se encuentra en el interior del paquete samples, el compilador califica el nombre de clase automáticamente en tiempo de compilación como su nombre completo: samples.SampleCode. El compilador también califica los nombres de propiedades y métodos, de forma que `sampleGreeting` y `sampleFunction()` se convierten en `samples.SampleCode.sampleGreeting` y `samples.SampleCode.sampleFunction()`, respectivamente.

Muchos desarrolladores, especialmente los que tienen experiencia en programación con Java, pueden elegir colocar únicamente clases en el nivel superior de un paquete. Sin embargo, ActionScript 3.0 no sólo admite clases en el nivel superior de un paquete, sino también variables, funciones e incluso sentencias. Un uso avanzado de esta característica consiste en definir un espacio de nombres en el nivel superior de un paquete de forma que esté disponible para todas las clases del paquete. Sin embargo, hay que tener en cuenta que sólo se admiten dos especificadores de acceso en el nivel superior de un paquete: `public` e `internal`. A diferencia de Java, que permite declarar clases anidadas como privadas, ActionScript 3.0 no admite clases anidadas privadas ni anidadas.

Sin embargo, en muchos otros aspectos los paquetes de ActionScript 3.0 son similares a los paquetes del lenguaje de programación Java. Como se puede ver en el ejemplo anterior, las referencias de nombre completo a paquetes se expresan con el operador punto (`.`), igual que en Java. Se pueden utilizar paquetes para organizar el código en una estructura jerárquica intuitiva que puedan usar otros programadores. Esto permite compartir código fácilmente, ya que ofrece una manera de crear un paquete para compartirlo con otros y de utilizar en el código paquetes creados por otros.

El uso de paquetes también ayuda a garantizar que los nombres de los identificadores utilizados son únicos y no entran en conflicto con otros nombres de identificador. De hecho, se podría decir que ésta es la ventaja principal de los paquetes. Por ejemplo, dos programadores que desean compartir código pueden haber creado una clase denominada SampleCode. Sin los paquetes, esto crearía un conflicto de nombres y la única resolución sería cambiar el nombre de una de las clases. Sin embargo, con los paquetes el conflicto de nombres se evita fácilmente colocando una de las clases (o las dos, preferiblemente) en paquetes con nombres únicos.

También se pueden incluir puntos incorporados en el nombre de paquete para crear paquetes anidados. Esto permite crear una organización jerárquica de paquetes. Un buen ejemplo de ello es el paquete flash.xml que proporciona ActionScript 3.0. Este paquete se anida dentro del paquete flash.

El paquete flash.xml contiene el analizador de XML antiguo, que se usaba en versiones anteriores de ActionScript. Una de las razones por las que ahora está en el paquete flash.xml es que hay un conflicto entre el nombre de la clase XML antigua y el nombre de la nueva clase XML que implementa la funcionalidad de la especificación de XML para ECMAScript (E4X) disponible en ActionScript 3.0.

Aunque pasar de la clase XML antigua a un paquete es un buen primer paso, la mayoría de los usuarios de las clases XML antiguas importarán el paquete flash.xml, lo que generará el mismo conflicto de nombres, a menos que recuerden que siempre deben utilizar el nombre completo de la clase XML antigua (flash.xml.XML). Para evitar esta situación, la clase XML antigua ahora se denomina XMLDocument, como se indica en el siguiente ejemplo:

```
package flash.xml
{
    class XMLDocument {}
    class XMLNode {}
    class XMLSocket {}
}
```

La mayor parte de ActionScript 3.0 se organiza en el paquete flash. Por ejemplo, el paquete flash.display contiene la API de la lista de visualización y el paquete flash.events contiene el nuevo modelo de eventos.

Creación de paquetes

ActionScript 3.0 proporciona una gran flexibilidad para organizar los paquetes, las clases y los archivos de código fuente. Las versiones anteriores de ActionScript sólo permitían una clase por archivo de código fuente y requerían que el nombre del archivo coincidiera con el nombre de la clase. ActionScript 3.0 permite incluir varias clases en un archivo de código fuente, pero sólo puede estar disponible una clase de cada archivo para el código externo a ese archivo. Es decir, sólo se puede declarar una clase de cada archivo en una declaración de paquete. Hay que declarar las clases adicionales fuera de la definición de paquete, lo que hace que estas clases sean invisibles para el código externo a ese archivo de código fuente. El nombre de clase declarado en la definición de paquete debe coincidir con el nombre del archivo de código fuente.

ActionScript 3.0 también proporciona más flexibilidad para la declaración de paquetes. En versiones anteriores de ActionScript, los paquetes sólo representaban directorios en los que se colocaban archivos de código fuente y no se declaraban con la sentencia `package`, sino que se incluía el nombre de paquete como parte del nombre completo de clase en la declaración de clase. Aunque los paquetes siguen representando directorios en ActionScript 3.0, pueden contener algo más que clases. En ActionScript 3.0 se utiliza la sentencia `package` para declarar un paquete, lo que significa que también se pueden declarar variables, funciones y espacios de nombres en el nivel superior de un paquete. Incluso se pueden incluir sentencias ejecutables en el nivel superior de un paquete. Si se declaran variables, funciones o espacios de nombres en el nivel superior de un paquete, los únicos atributos disponibles en ese nivel son `public` e `internal`, y sólo una declaración de nivel de paquete por archivo puede utilizar el atributo `public`, independientemente de que la declaración sea una clase, una variable, una función o un espacio de nombres.

Los paquetes son útiles para organizar el código y para evitar conflictos de nombres. No se debe confundir el concepto de paquete con el concepto de herencia de clases, que no está relacionado con el anterior. Dos clases que residen en el mismo paquete tendrán un espacio de nombres común, pero no estarán necesariamente relacionadas entre sí de ninguna otra manera. Asimismo, un paquete anidado puede no tener ninguna relación semántica con su paquete principal.

Importación de paquetes

Si se desea utilizar una clase que está dentro un paquete, se debe importar el paquete o la clase específica. Esto varía con respecto a ActionScript 2.0, donde la importación de clases era opcional.

Por ejemplo, considérese el ejemplo de la clase SampleCode mencionado antes en este capítulo. Si la clase reside en un paquete denominado samples, hay que utilizar una de las siguientes sentencias de importación antes de utilizar la clase SampleCode:

```
import samples.*;
```

o

```
import samples.SampleCode;
```

En general, la sentencias `import` deben ser lo más específicas posible. Si se pretende utilizar la clase SampleCode desde el paquete samples, hay que importar únicamente la clase SampleCode, no todo el paquete al que pertenece. La importación de paquetes completos puede producir conflictos de nombres inesperados.

También se debe colocar el código fuente que define el paquete o la clase en la *ruta de clases*. La ruta de clases es una lista de rutas de directorio locales definida por el usuario que determina dónde buscará el compilador los paquetes y las clases importados. La ruta de clases se denomina a veces *ruta de compilación* o *ruta de código fuente*.

Tras importar correctamente la clase o el paquete, se puede utilizar el nombre completo de la clase (samples.SampleCode) o simplemente el nombre de clase (SampleCode).

Los nombres completos son útiles cuando hay ambigüedad en el código a causa de clases, métodos o propiedades con nombre idénticos, pero pueden ser difíciles de administrar si se usan para todos los identificadores. Por ejemplo, la utilización del nombre completo produce código demasiado extenso al crear una instancia de la clase SampleCode:

```
var mySample:samples.SampleCode = new samples.SampleCode();
```

A medida que aumentan los niveles de paquetes anidados, la legibilidad del código disminuye. En las situaciones en las que se esté seguro de que los identificadores ambiguos no constituirán un problema, se puede hacer el código más legible utilizando identificadores sencillos. Por ejemplo, al crear una nueva instancia de la clase SampleCode, el resultado será mucho menos extenso si se utiliza sólo el identificador de clase:

```
var mySample:SampleCode = new SampleCode();
```

Si se intenta utilizar nombres de identificador sin importar primero el paquete o la clase apropiados, el compilador no podrá encontrar las definiciones de clase. Por otra parte, si se importa un paquete o una clase, cualquier intento de definir un nombre que entre en conflicto con un nombre importado generará un error.

Cuando se crea un paquete, el especificador de acceso predeterminado para todos los miembros del paquete es `internal`, lo que significa que, de manera predeterminada, los miembros del paquete sólo estarán visibles para los otros miembros del paquete. Si se desea que una clase esté disponible para código externo al paquete, se debe declarar la clase como `public`. Por ejemplo, el siguiente paquete contiene dos clases, SampleCode y CodeFormatter:

```
// SampleCode.as file
package samples
{
    public class SampleCode {}
}

// CodeFormatter.as file
package samples
{
    class CodeFormatter {}
}
```

La clase SampleCode está visible fuera del paquete porque se ha declarado como una clase `public`. Sin embargo, la clase CodeFormatter sólo está visible dentro del mismo paquete samples. Si se intenta acceder la clase CodeFormatter fuera del paquete samples, se generará un error, como se indica en el siguiente ejemplo:

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // okay, public class
var myFormatter:CodeFormatter = new CodeFormatter(); // error
```

Si se desea que ambas clases estén disponibles para código externo al paquete, se deben declarar las dos como `public`. No se puede aplicar el atributo `public` a la declaración del paquete.

Los nombres completos son útiles para resolver conflictos de nombres que pueden producirse al utilizar paquetes. Este escenario puede surgir si se importan dos paquetes que definen clases con el mismo identificador. Por ejemplo, considérese el siguiente paquete, que también tiene una clase denominada SampleCode:

```
package langref.samples
{
    public class SampleCode {}
}
```

Si se importan ambas clases de la manera siguiente, se producirá un conflicto de nombres al hacer referencia a la clase SampleCode:

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // name conflict
```

El compilador no sabe qué clase SampleCode debe utilizar. Para resolver este conflicto, hay que utilizar el nombre completo de cada clase, de la manera siguiente:

```
var sample1:samples.SampleCode = new samples.SampleCode();
var sample2:langref.samples.SampleCode = new langref.samples.SampleCode();
```

Nota: *los programadores con experiencia en C++ suelen confundir la sentencia* `import` *con* `#include`*. La directiva* `#include` *es necesaria en C++ porque los compiladores de C++ procesan un archivo cada vez y no buscan definiciones de clases en otros archivos, a menos que se incluya explícitamente un archivo de encabezado. ActionScript 3.0 tiene una directiva* `include`*, pero no está diseñada para importar clases y paquetes. Para importar clases o paquetes en ActionScript 3.0, hay que usar la sentencia* `import` *y colocar el archivo de código fuente que contiene el paquete en la ruta de clases.*

Espacios de nombres

Los espacios de nombres ofrecen control sobre la visibilidad de las propiedades y los métodos que se creen. Los especificadores de control de acceso `public`, `private`, `protected` e `internal` son espacios de nombres incorporados. Si estos especificadores de control de acceso predefinidos no se adaptan a las necesidades del programador, es posible crear espacios de nombres personalizados.

Para los programadores que estén familiarizados con los espacios de nombres XML, gran parte de lo que se va a explicar no resultará nuevo, aunque la sintaxis y los detalles de la implementación de ActionScript son ligeramente distintos de los de XML. Si nunca se ha trabajado con espacios de nombres, el concepto en sí es sencillo, pero hay que aprender la terminología específica de la implementación.

Para comprender mejor cómo funcionan los espacios de nombres, es necesario saber que el nombre de una propiedad o un método siempre contiene dos partes: un identificador y un espacio de nombres. El identificador es lo que generalmente se considera un nombre. Por ejemplo, los identificadores de la siguiente definición de clase son `sampleGreeting` y `sampleFunction()`:

```
class SampleCode
{
    var sampleGreeting:String;
    function sampleFunction () {
        trace(sampleGreeting + " from sampleFunction()");
    }
}
```

Siempre que las definiciones no estén precedidas por un atributo de espacio de nombres, sus nombres se califican mediante el espacio de nombres `internal` predeterminado, lo que significa que sólo estarán visibles para los orígenes de llamada del mismo paquete. Si el compilador está configurado en modo estricto, emite una advertencia para indicar que el espacio de nombres `internal` se aplica a cualquier identificador que no tenga un atributo de espacio de nombres. Para asegurarse de que un identificador esté disponible en todas partes, hay que usar específicamente el atributo `public` como prefijo del nombre de identificador. En el ejemplo de código anterior, `sampleGreeting` y `sampleFunction()` tienen `internal` como valor de espacio de nombres.

Se deben seguir tres pasos básicos al utilizar espacios de nombres. En primer lugar, hay que definir el espacio de nombres con la palabra clave `namespace`. Por ejemplo, el código siguiente define el espacio de nombres `version1`:

```
namespace version1;
```

En segundo lugar, se aplica el espacio de nombres utilizándolo en lugar de utilizar un especificador de control de acceso en una declaración de propiedad o método. El ejemplo siguiente coloca una función denominada `myFunction()` en el espacio de nombres `version1`:

```
version1 function myFunction() {}
```

En tercer lugar, tras aplicar el espacio de nombres, se puede hacer referencia al mismo con la directiva `use` o calificando el nombre de un identificador con un espacio de nombres. En el ejemplo siguiente se hace referencia a la función `myFunction()` mediante la directiva `use`:

```
use namespace version1;
myFunction();
```

También se puede utilizar un nombre completo para hacer referencia a la función `myFunction()`, como se indica en el siguiente ejemplo:

```
version1::myFunction();
```

Definición de espacios de nombres

Los espacios de nombres contienen un valor, el Identificador uniforme de recurso (URI), que a veces se denomina *nombre del espacio de nombres*. El URI permite asegurarse de que la definición del espacio de nombres es única.

Para crear un espacio de nombres se declara una definición de espacio de nombres de una de las dos maneras siguientes. Se puede definir un espacio de nombres con un URI explícito, de la misma manera que se define un espacio de nombres XML, o se puede omitir el URI. En el siguiente ejemplo se muestra la manera de definir un espacio de nombres mediante un URI:

```
namespace flash_proxy = "http://www.adobe.com/flash/proxy";
```

El URI constituye una cadena de identificación única para ese espacio de nombres. Si se omite el URI, como en el siguiente ejemplo, el compilador creará una cadena de identificación interna única en lugar del URI. El usuario no tiene acceso a esta cadena de identificación interna.

```
namespace flash_proxy;
```

Una vez definido un espacio de nombres, con o sin URI, ese espacio de nombres no podrá definirse de nuevo en el mismo ámbito. Si se intenta definir un espacio de nombres definido previamente en el mismo ámbito, se producirá un error del compilador.

Si se define un espacio de nombres dentro de un paquete o una clase, el espacio de nombres puede no estar visible para código externo al paquete o la clase, a menos que se use el especificador de control de acceso apropiado. Por ejemplo, el código siguiente muestra el espacio de nombres `flash_proxy` definido en el paquete flash.utils. En el siguiente ejemplo, la falta de un especificador de control de acceso significa que el espacio de nombres `flash_proxy` sólo estará visible para el código del paquete flash.utils y no estará visible para el código externo al paquete:

```
package flash.utils
{
    namespace flash_proxy;
}
```

El código siguiente utiliza el atributo `public` para hacer que el espacio de nombres `flash_proxy` esté visible para el código externo al paquete:

```
package flash.utils
{
    public namespace flash_proxy;
}
```

Aplicación de espacios de nombres

Aplicar un espacio de nombres significa colocar una definición en un espacio de nombres. Entre las definiciones que se puede colocar en espacios de nombres se incluyen funciones, variables y constantes (no se puede colocar una clase en un espacio de nombres personalizado).

Considérese, por ejemplo, una función declarada con el espacio de nombres de control de acceso `public`. Al utilizar el atributo `public` en una definición de función se coloca la función en el espacio de nombres public, lo que hace que esté disponible para todo el código. Una vez definido un espacio de nombres, se puede utilizar el espacio de nombres definido de la misma manera que se utilizaría el atributo `public` y la definición estará disponible para el código que puede hacer referencia al espacio de nombres personalizado. Por ejemplo, si se define un espacio de nombres `example1`, se puede añadir un método denominado `myFunction()` utilizando `example1` como un atributo, como se indica en el siguiente ejemplo:

```
namespace example1;
class someClass
{
    example1 myFunction() {}
}
```

Si se declara el método `myFunction()` con el espacio de nombres `example1` como un atributo, significa que el método pertenece al espacio de nombres `example1`.

Se debe tener en cuenta lo siguiente al aplicar espacios de nombres:

- Sólo se puede aplicar un espacio de nombres a cada declaración.
- No hay manera de aplicar un atributo de espacio de nombres a más de una definición simultáneamente. Es decir, si se desea aplicar el espacio de nombres a diez funciones distintas, se debe añadir el espacio de nombres como un atributo a cada una de las diez definiciones de función.
- Si se aplica un espacio de nombres, no se puede especificar también un especificador de control de acceso, ya que los espacios de nombres y los especificadores de control de acceso son mutuamente excluyentes. Es decir, no se puede declarar una función o propiedad como `public`, `private`, `protected` o `internal` y aplicar también el espacio de nombres.

Referencia a un espacio de nombres

No es necesario hacer referencia explícita a un espacio de nombres al usar un método o una propiedad declarados con alguno de los espacios de nombres de control de acceso, como `public`, `private`, `protected` e `internal`. Esto se debe a que el acceso a estos espacios de nombres especiales se controla por el contexto. Por ejemplo, las definiciones colocadas en el espacio de nombres `private` estarán disponibles automáticamente para el código de la misma clase. Sin embargo, para los espacios de nombres personalizados que se definan no existirá este control mediante el contexto. Para poder utilizar un método o una propiedad que se ha colocado en un espacio de nombres personalizado, se debe hacer referencia al espacio de nombres.

Se puede hacer referencia a espacios de nombres con la directiva `use namespace` o se puede calificar el nombre con el espacio de nombres mediante el signo calificador de nombre (`::`). Al hacer referencia a un espacio de nombres con la directiva `use namespace` "se abre" el espacio de nombres, de forma que se puede aplicar a cualquier identificador que no esté calificado. Por ejemplo, si se define el espacio de nombres `example1`, se puede acceder a nombres de ese espacio de nombres mediante `use namespace example1`:

```
use namespace example1;
myFunction();
```

Es posible abrir más de un espacio de nombres simultáneamente. Cuando se abre un espacio de nombres con `use namespace`, permanece abierto para todo el bloque de código en el que se abrió. No hay forma de cerrar un espacio de nombres explícitamente.

Sin embargo, si hay más de un espacio de nombres abierto, aumenta la probabilidad de que se produzcan conflictos de nombres. Si se prefiere no abrir un espacio de nombres, se puede evitar la directiva `use namespace` calificando el nombre del método o la propiedad con el espacio de nombres y el signo calificador de nombre. Por ejemplo, el código siguiente muestra la manera de calificar el nombre `myFunction()` con el espacio de nombres `example1`:

```
example1::myFunction();
```

Utilización de espacios de nombres

Puede encontrar un ejemplo real de un espacio de nombres que se utiliza para evitar conflictos de nombre en la clase flash.utils.Proxy que forma parte de ActionScript 3.0. La clase Proxy, que es la sustitución de la propiedad `Object.__resolve` de ActionScript 2.0, permite interceptar referencias a propiedades o métodos no definidos antes de que se produzca un error. Todos los métodos de la clase Proxy residen en el espacio de nombres `flash_proxy` para evitar conflictos de nombres.

Para entender mejor cómo se utiliza el espacio de nombres `flash_proxy`, hay que entender cómo se utiliza la clase Proxy. La funcionalidad de la clase Proxy sólo está disponible para las clases que heredan de ella. Es decir, si se desea utilizar los métodos de la clase Proxy en un objeto, la definición de clase del objeto debe ampliar la clase Proxy. Por ejemplo, si desea se interceptar intentos de llamar a un método no definido, se debe ampliar la clase Proxy y después reemplazar el método `callProperty()` de la clase Proxy.

La implementación de espacios de nombres es generalmente un proceso en tres pasos consistente en definir y aplicar un espacio de nombres, y después hacer referencia al mismo. Sin embargo, como nunca se llama explícitamente a ninguno de los métodos de la clase Proxy, sólo se define y aplica el espacio de nombres `flash_proxy`, pero nunca se hace referencia al mismo. ActionScript 3.0 define el espacio de nombres `flash_proxy` y lo aplica en la clase Proxy. El código sólo tiene que aplicar el espacio de nombres `flash_proxy` a las clases que amplían la clase Proxy.

El espacio de nombres `flash_proxy` se define en el paquete flash.utils de una manera similar a la siguiente:

```
package flash.utils
{
    public namespace flash_proxy;
}
```

El espacio de nombres se aplica a los métodos de la clase Proxy, como se indica en el siguiente fragmento de dicha clase:

```
public class Proxy
{
    flash_proxy function callProperty(name:*, ... rest):*
    flash_proxy function deleteProperty(name:*):Boolean
    ...
}
```

Como se indica en el código siguiente, se debe importar primero la clase Proxy y el espacio de nombres `flash_proxy`. A continuación se debe declarar la clase de forma que amplíe la clase Proxy (también se debe añadir el atributo `dynamic` si se compila en modo estricto). Al sustituir el método `callProperty()`, se debe utilizar el espacio de nombres `flash_proxy`.

```
package
{
    import flash.utils.Proxy;
    import flash.utils.flash_proxy;

    dynamic class MyProxy extends Proxy
    {
        flash_proxy override function callProperty(name:*, ...rest):*
        {
            trace("method call intercepted: " + name);
        }
    }
}
```

Si se crea una instancia de la clase MyProxy y se llama a un método no definido, como el método `testing()` llamado en el ejemplo siguiente, el objeto Proxy intercepta la llamada al método y ejecuta las sentencias del método `callProperty()` sustituido (en este caso, una simple sentencia `trace()`).

```
var mySample:MyProxy = new MyProxy();
mySample.testing(); // method call intercepted: testing
```

Tener los métodos de la clase Proxy dentro del espacio de nombres `flash_proxy` ofrece dos ventajas. En primer lugar, tener un espacio de nombres independiente reduce el desorden en la interfaz pública de cualquier clase que amplíe la clase Proxy. (Hay aproximadamente una docena de métodos en la clase Proxy que se pueden sustituir; todos ellos han sido diseñados para no ser llamados directamente. Colocarlos todos en el espacio de nombres public podría provocar confusiones.) En segundo lugar, el uso del espacio de nombres `flash_proxy` evita los conflictos de nombres en caso de que la subclase de Proxy contenga métodos de instancia con nombres que coincidan con los de los métodos de la clase Proxy. Por ejemplo, supongamos que un programador desea asignar a uno de sus métodos el nombre `callProperty()`. El código siguiente es aceptable porque su versión del método `callProperty()` está en un espacio de nombres distinto:

```
dynamic class MyProxy extends Proxy
{
    public function callProperty() {}
    flash_proxy override function callProperty(name:*, ...rest):*
    {
        trace("method call intercepted: " + name);
    }
}
```

Los espacios de nombres también pueden ser útiles cuando se desea proporcionar acceso a métodos o propiedades de una manera que se no puede realizar con los cuatro especificadores de control de acceso (`public`, `private`, `internal` y `protected`). Por ejemplo, un programador puede tener algunos métodos de utilidad repartidos por varios paquetes. Desea que estos métodos estén disponibles para todos los paquetes, pero no quiere que sean públicos. Para ello, se puede crear un nuevo espacio de nombres y utilizarlo como un especificador de control de acceso especial.

En el ejemplo siguiente se utiliza un espacio de nombres definido por el usuario para agrupar dos funciones que residen en paquetes distintos. Al agruparlos en el mismo espacio de nombres, se puede hacer que ambas funciones estén visibles para una clase o un paquete mediante una única sentencia `use namespace`.

En este ejemplo se utilizan cuatro archivos para ilustrar la técnica. Todos los archivos deben estar en la ruta de clases. El primer archivo, myInternal.as, se utiliza para definir el espacio de nombres `myInternal`. Como el archivo está en un paquete denominado example, se debe colocar en una carpeta denominada example. El espacio de nombres se marca como `public` de forma que se pueda importar en otros paquetes.

```
// myInternal.as in folder example
package example
{
    public namespace myInternal = "http://www.adobe.com/2006/actionscript/examples";
}
```

Los archivos Utility.as y Helper.as definen las clases que contienen los métodos que deben estar disponibles para otros paquetes. La clase Utility está en el paquete example.alpha, lo que significa que se debe colocar el archivo dentro de una subcarpeta de la carpeta example denominada alpha. La clase Helper está en el paquete example.beta, por lo que se debe colocar el archivo dentro de otra subcarpeta de la carpeta example denominada beta. Ambos paquetes, example.alpha y example.beta, deben importar el espacio de nombres antes de utilizarlo.

```
// Utility.as in the example/alpha folder
package example.alpha
{
    import example.myInternal;

    public class Utility
    {
        private static var _taskCounter:int = 0;

        public static function someTask()
        {
            _taskCounter++;
        }

        myInternal static function get taskCounter():int
        {
            return _taskCounter;
        }
    }
}

// Helper.as in the example/beta folder
package example.beta
{
    import example.myInternal;

    public class Helper
    {
        private static var _timeStamp:Date;

        public static function someTask()
        {
            _timeStamp = new Date();
        }

        myInternal static function get lastCalled():Date
        {
            return _timeStamp;
        }
    }
}
```

El cuarto archivo, NamespaceUseCase.as, es la clase principal de la aplicación, y debe estar en el mismo nivel que la carpeta example. En Adobe Flash CS4 Professional, esta clase se utilizaría como la clase de documento para el archivo FLA. La clase NamespaceUseCase también importa el espacio de nombres `myInternal` y lo utiliza para llamar a los dos métodos estáticos que residen en los otros paquetes. El ejemplo utiliza métodos estáticos sólo para simplificar el código. Se pueden colocar tanto métodos estáticos como métodos de instancia en el espacio de nombres `myInternal`.

```
// NamespaceUseCase.as
package
{
    import flash.display.MovieClip;
    import example.myInternal; // import namespace
    import example.alpha.Utility;// import Utility class
    import example.beta.Helper;// import Helper class

    public class NamespaceUseCase extends MovieClip
    {
        public function NamespaceUseCase()
        {
            use namespace myInternal;

            Utility.someTask();
            Utility.someTask();
            trace(Utility.taskCounter); // 2

            Helper.someTask();
            trace(Helper.lastCalled); // [time someTask() was last called]
        }
    }
}
```

Variables

Las variables permiten almacenar los valores que se utilizan en el programa. Para declarar una variable se debe utilizar la sentencia `var` con el nombre de variable. En ActionScript 2.0, el uso de la sentencia `var` sólo es necesario si se usan anotaciones de tipo de datos. En ActionScript 3.0, el uso de la sentencia `var` es siempre obligatorio. Por ejemplo, la línea siguiente de código ActionScript declara una variable denominada `i`:

```
var i;
```

Si se omite la sentencia `var` al declarar una variable, se producirá un error del compilador en modo estricto y un error en tiempo de ejecución en modo estándar. Por ejemplo, la siguiente línea de código producirá un error si la variable `i` no se ha definido previamente:

```
i; // error if i was not previously defined
```

La asociación de una variable con un tipo de datos, debe realizarse al declarar la variable. Es posible declarar una variable sin designar su tipo, pero esto generará una advertencia del compilador en modo estricto. Para designar el tipo de una variable se añade el nombre de la variable con un signo de dos puntos (:), seguido del tipo de la variable. Por ejemplo, el código siguiente declara una variable `i` de tipo int:

```
var i:int;
```

Se puede asignar un valor a una variable mediante el operador de asignación (=). Por ejemplo, el código siguiente declara una variable `i` y le asigna el valor 20:

```
var i:int;
i = 20;
```

Puede ser más cómodo asignar un valor a una variable a la vez que se declara la variable, como en el siguiente ejemplo:

```
var i:int = 20;
```

La técnica de asignar un valor a una variable en el momento de declararla se suele utilizar no sólo al asignar valores simples, como enteros y cadenas, sino también al crear un conjunto o una instancia de una clase. En el siguiente ejemplo se muestra un conjunto que se declara y recibe un valor en una línea de código.

```
var numArray:Array = ["zero", "one", "two"];
```

Para crear una instancia de una clase hay que utilizar el operador `new`. En el ejemplo siguiente se crea una instancia de una clase denominada `CustomClass` y se asigna una referencia a la instancia de clase recién creada a la variable denominada `customItem`:

```
var customItem:CustomClass = new CustomClass();
```

Si hubiera que declarar más de una variable, se pueden declarar todas en una línea de código utilizando el operador coma (`,`) para separar las variables. Por ejemplo, el código siguiente declara tres variables en una línea de código:

```
var a:int, b:int, c:int;
```

También se puede asignar valores a cada una de las variables en la misma línea de código. Por ejemplo, el código siguiente declara tres variables (`a`, `b` y `c`) y asigna un valor a cada una de ellas:

```
var a:int = 10, b:int = 20, c:int = 30;
```

Aunque se puede utilizar el operador coma para agrupar declaraciones de variables en una sentencia, al hacerlo el código será menos legible.

Aspectos básicos del ámbito de variables

El *ámbito* de una variable es el área del código en la que se puede acceder a la variable mediante una referencia léxica. Una variable *global* está definida en todas las áreas del código, mientras que una variable *local* sólo está definida en una parte del código. En ActionScript 3.0, las variables siempre se asignan al ámbito de la función o la clase en la que se han declarado. Una variable global es una variable que se define fuera de una definición de función o clase. Por ejemplo, el código siguiente crea una variable global `strGlobal` declarándola fuera de las funciones. El ejemplo muestra que una variable global está disponible tanto dentro como fuera de una definición de función.

```
var strGlobal:String = "Global";
function scopeTest()
{
    trace(strGlobal); // Global
}
scopeTest();
trace(strGlobal); // Global
```

Las variables locales se declaran dentro de una definición de función. La parte más pequeña de código para la que se puede definir una variable local es una definición de función. Una variable local declarada en una función sólo existirá en esa función. Por ejemplo, si se declara una variable denominada `str2` dentro de una función denominada `localScope()`, dicha variable no estará disponible fuera de la función.

```
function localScope()
{
    var strLocal:String = "local";
}
localScope();
trace(strLocal); // error because strLocal is not defined globally
```

Si el nombre de la variable que utiliza para la variable local ya se ha declarado como variable global, la definición local oculta (o reemplaza) la definición global mientras la variable local se encuentre dentro del ámbito. La variable global continuará existiendo fuera de la función. Por ejemplo, el siguiente fragmento de código crea una variable de cadena global denominada `str1` y a continuación crea una variable local con el mismo nombre dentro de la función `scopeTest()`. La sentencia `trace` de la función devuelve el valor local de la variable, pero la sentencia `trace` situada fuera de la función devuelve el valor global de la variable.

```
var str1:String = "Global";
function scopeTest ()
{
    var str1:String = "Local";
    trace(str1); // Local
}
scopeTest();
trace(str1); // Global
```

Las variables de ActionScript, a diferencia de las variables de C++ y Java, no tienen ámbito a nivel de bloque. Un bloque de código es un grupo de sentencias entre una llave inicial ({) y una llave final (}). En algunos lenguajes de programación, como C++ y Java, las variables declaradas dentro de un bloque de código no están disponibles fuera de ese bloque de código. Esta restricción de ámbito se denomina ámbito a nivel de bloque y no existe en ActionScript. Si se declara una variable dentro de un bloque de código, dicha variable no sólo estará disponible en ese bloque de código, sino también en cualquier otra parte de la función a la que pertenece el bloque de código. Por ejemplo, la siguiente función contiene variables definidas en varios ámbitos de bloque. Todas las variables están disponibles en toda la función.

```
function blockTest (testArray:Array)
{
    var numElements:int = testArray.length;
    if (numElements > 0)
    {
        var elemStr:String = "Element #";
        for (var i:int = 0; i < numElements; i++)
        {
            var valueStr:String = i + ": " + testArray[i];
            trace(elemStr + valueStr);
        }
        trace(elemStr, valueStr, i); // all still defined
    }
    trace(elemStr, valueStr, i); // all defined if numElements > 0
}

blockTest(["Earth", "Moon", "Sun"]);
```

Una consecuencia interesante de la falta de ámbito a nivel de bloque es que se puede leer el valor de una variable o escribir en ella antes de que se declare, con tal de que se declare antes del final de la función. Esto se debe a una técnica denominada *hoisting*, que consiste en que el compilador mueve todas las declaraciones de variables al principio de la función. Por ejemplo, el código siguiente se compila aunque la función `trace()` inicial para la variable `num` está antes de la declaración de la variable `num`:

```
trace(num); // NaN
var num:Number = 10;
trace(num); // 10
```

Sin embargo, el compilador no mueve ninguna sentencia de asignación al principio. Esto explica por qué la función `trace()` inicial de `num` devuelve `NaN` (no es un número), que es el valor predeterminado para variables con tipo de datos Number. Así, es posible asignar valores a variables incluso antes de declararlas, como se muestra en el siguiente ejemplo:

```
num = 5;
trace(num); // 5
var num:Number = 10;
trace(num); // 10
```

Valores predeterminados

Un *valor predeterminado* es el valor que contiene una variable antes de que se establezca su valor. Una variable se *inicializa* al establecer su valor por primera vez. Si se declara una variable pero no establece su valor, dicha variable estará *sin inicializar*. El valor de una variable no inicializada depende del tipo de datos que tenga. En la tabla siguiente se describen los valores predeterminados de las variables, clasificados por tipo de datos:

Tipo de datos	**Valor predeterminado**
Boolean	`false`
int	0
Number	`NaN`
Object	`null`
String	null
uint	0
No declarada (equivalente a anotación de tipo `*`)	`undefined`
Todas las demás clases, incluidas las clases definidas por el usuario.	`null`

Para variables de tipo Number, el valor predeterminado es `NaN` (no es un número), que es un valor especial definido por la norma IEEE-754 para designar un valor que no representa un número.

Si se declara una variable pero no se declara su tipo de datos, se aplica el tipo de datos predeterminado, `*`, que en realidad indica que la variable no tiene tipo. Si tampoco se inicializa una variable sin tipo con un valor, su valor predeterminado será `undefined`.

Para tipos de datos distintos de Boolean, Number, int y uint, el valor predeterminado de cualquier variable no inicializada es `null`. Esto se aplica a todas las clases definidas mediante ActionScript 3.0, así como a las clases personalizadas que se creen.

El valor `null` no es válido para variables de tipo Boolean, Number, int o uint. Si se intenta asignar el valor `null` a una variable de este tipo, dicho valor se convierte en el valor predeterminado para ese tipo de datos. A las variables de tipo Object se les puede asignar un valor `null`. Si se intenta asignar el valor `undefined` a una variable de tipo Object, dicho valor se convierte en `null`.

Para variables de tipo Number, hay una función especial de nivel superior denominada `isNaN()` que devuelve el valor booleano `true` si la variable no es un número, y `false` en caso contrario.

Tipos de datos

Un *tipo de datos* define un conjunto de valores. Por ejemplo, el tipo de datos Boolean es el conjunto de dos valores bien definidos: `true` y `false`. Además del tipo de datos Boolean, ActionScript 3.0 define varios tipos de datos utilizados con frecuencia, como String, Number y Array. Un programador puede definir sus propios tipos de datos utilizando clases o interfaces para definir un conjunto de valores personalizado. En ActionScript 3.0 todos los valores, tanto simples como complejos, son objetos.

Un *valor simple* es un valor que pertenece a uno de los tipos de datos siguientes: Boolean, int, Number, String y uint. Trabajar con valores simples suele ser más rápido que trabajar con valores complejos, ya que ActionScript almacena los valores simples de una manera especial que permite optimizar la velocidad y el uso de la memoria.

Nota: *para los interesados en los detalles técnicos, ActionScript almacena internamente los valores simples como objetos inmutables. El hecho de que se almacenen como objetos inmutables significa que pasar por referencia es en realidad lo mismo que pasar por valor. Esto reduce el uso de memoria y aumenta la velocidad de ejecución, ya que las referencias ocupan normalmente bastante menos que los valores en sí.*

Un *valor complejo* es un valor que no es un valor simple. Entre los tipos de datos que definen conjuntos de valores complejos se encuentran Array, Date, Error, Function, RegExp, XML y XMLList.

Muchos lenguajes de programación distinguen entre los valores simples y los objetos que los contienen. Java, por ejemplo, tiene un valor simple int y la clase java.lang.Integer que lo contiene. Los valores simples de Java no son objetos, pero los objetos que los contienen sí, lo que hace que los valores simples sean útiles para algunas operaciones y los objetos contenedores sean más adecuados para otras operaciones. En ActionScript 3.0, los valores simples y sus objetos contenedores son, para fines prácticos, indistinguibles. Todos los valores, incluso los valores simples, son objetos. Flash Player y Adobe AIR tratan estos tipos simples como casos especiales que se comportan como objetos pero que no requieren la sobrecarga normal asociada a la creación de objetos. Esto significa que las dos líneas de código siguientes son equivalentes:

```
var someInt:int = 3;
var someInt:int = new int(3);
```

Todos los tipos de datos simples y complejos antes descritos se definen en las clases principales de ActionScript 3.0. Las clases principales permiten crear objetos utilizando valores literales en lugar de utilizar el operador `new`. Por ejemplo, se puede crear un conjunto utilizando un valor literal o el constructor de la clase Array, de la manera siguiente:

```
var someArray:Array = [1, 2, 3]; // literal value
var someArray:Array = new Array(1,2,3); // Array constructor
```

Verificación de tipos

La verificación de tipos puede tener lugar al compilar o en tiempo de ejecución. Los lenguajes con tipos estáticos, como C++ y Java, realizan la verificación de tipos en tiempo de compilación. Los lenguajes con tipos dinámicos, como Smalltalk y Python, realizan la verificación de tipos en tiempo de ejecución. Al ser un lenguaje con tipos dinámicos, ActionScript 3.0 ofrece verificación de tipos en tiempo de ejecución, pero también admite verificación de tipos en tiempo de compilación con un modo de compilador especial denominado *modo estricto*. En modo estricto, la verificación de tipos se realiza en tiempo de compilación y en tiempo de ejecución; en cambio, en modo estándar la verificación de tipos sólo se realiza en tiempo de ejecución.

Los lenguajes con tipos dinámicos ofrecen una gran flexibilidad para estructurar el código, pero a costa de permitir que se produzcan errores de tipo en tiempo de ejecución. Los lenguajes con tipos estáticos notifican los errores de tipo en tiempo de compilación, pero a cambio deben conocer la información de tipos en tiempo de compilación.

Verificación de tipos en tiempo de compilación

La verificación de tipos en tiempo de compilación se suele favorecer en los proyectos grandes ya que, a medida que el tamaño de un proyecto crece, la flexibilidad de los tipos de datos suele ser menos importante que detectar los errores de tipo lo antes posible. Ésta es la razón por la que, de manera predeterminada, el compilador de ActionScript de Adobe Flash CS4 Professional y Adobe Flex Builder está configurado para ejecutarse en modo estricto.

Para poder proporcionar verificación de tipos en tiempo de compilación, el compilador necesita saber cuáles son los tipos de datos de las variables o las expresiones del código. Para declarar explícitamente un tipo de datos para una variable, se debe añadir el operador dos puntos (`:`) seguido del tipo de datos como sufijo del nombre de la variable. Para asociar un tipo de datos a un parámetro, se debe utilizar el operador dos puntos seguido del tipo de datos. Por ejemplo, el código siguiente añade la información de tipo de datos al parámetro `xParam` y declara una variable `myParam` con un tipo de datos explícito:

```
function runtimeTest(xParam:String)
{
    trace(xParam);
}
var myParam:String = "hello";
runtimeTest(myParam);
```

En modo estricto, el compilador de ActionScript notifica los tipos no coincidentes como errores de compilación. Por ejemplo, el código siguiente declara un parámetro de función `xParam`, de tipo Object, pero después intenta asignar valores de tipo String y Number a ese parámetro. Esto produce un error del compilador en modo estricto.

```
function dynamicTest(xParam:Object)
{
    if (xParam is String)
    {
        var myStr:String = xParam; // compiler error in strict mode
        trace("String: " + myStr);
    }
    else if (xParam is Number)
    {
        var myNum:Number = xParam; // compiler error in strict mode
        trace("Number: " + myNum);
    }
}
```

Sin embargo, incluso en modo estricto se puede evitar selectivamente la verificación de tipos en tiempo de compilación dejando sin tipo el lado derecho de una sentencia de asignación. También se puede marcar una variable o expresión como variable o expresión sin tipo, omitiendo una anotación de tipo o utilizando la anotación de tipo asterisco (`*`). Por ejemplo, si se modifica el parámetro `xParam` del ejemplo anterior de forma que ya no tenga una anotación de tipo, el código se compilará en modo estricto:

```
function dynamicTest(xParam)
{
    if (xParam is String)
    {
        var myStr:String = xParam;
        trace("String: " + myStr);
    }
    else if (xParam is Number)
    {
        var myNum:Number = xParam;
        trace("Number: " + myNum);
    }
}
dynamicTest(100)
dynamicTest("one hundred");
```

Verificación de tipos en tiempo de ejecución

ActionScript 3.0 realiza la verificación de tipos en tiempo de ejecución tanto si se compila en modo estricto como si se compila en modo estándar. Considérese una situación en la que se pasa el valor 3 como argumento a una función que espera un conjunto. En modo estricto, el compilador generará un error, ya que el valor 3 no es compatible con el tipo de datos Array. Al desactivar el modo estricto y entrar en modo estándar, el compilador no notificará acerca de los tipos no coincidentes, pero la verificación de tipos en tiempo de ejecución realizada por Flash Player y Adobe AIR dará lugar a un error en tiempo de ejecución.

En el siguiente ejemplo se muestra una función denominada `typeTest()` que espera un argumento de tipo Array pero recibe el valor 3. Esto provoca un error en tiempo de ejecución en modo estándar, ya que el valor 3 no es un miembro del tipo de datos (Array) declarado para el parámetro.

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum:Number = 3;
typeTest(myNum);
// run-time error in ActionScript 3.0 standard mode
```

También puede haber situaciones en las que se produzca un error de tipo en tiempo de ejecución aunque se opere en modo estricto. Esto puede suceder si se usa el modo estricto pero se elige no verificar tipos en tiempo de compilación utilizando una variable sin tipo. Si se utiliza una variable sin tipo, no se elimina la verificación de tipos, sino que se aplaza hasta el tiempo de ejecución. Por ejemplo, si la variable `myNum` del ejemplo anterior no tiene un tipo de datos declarado, el compilador no podrá detectar los tipos no coincidentes, pero Flash Player y Adobe AIR generarán un error en tiempo de ejecución al comparar el valor en tiempo de ejecución de `myNum`, que está definido en 3 como resultado de la sentencia de asignación, con el tipo de `xParam`, que está definido como el tipo de datos Array.

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum = 3;
typeTest(myNum);
// run-time error in ActionScript 3.0
```

La verificación de tipos en tiempo de ejecución también permite un uso más flexible de la herencia que la verificación en tiempo de compilación. Al aplazar la verificación de tipos al tiempo de ejecución, el modo estándar permite hacer referencia a propiedades de una subclase aunque se realice una *conversión hacia arriba*. Una conversión hacia arriba se produce si se utiliza una clase base para declarar el tipo de una instancia de la clase y una subclase para crear la instancia. Por ejemplo, se puede crear una clase denominada ClassBase ampliable (las clases con el atributo `final` no se pueden ampliar):

```
class ClassBase
{
}
```

Posteriormente se puede crear una subclase de ClassBase denominada ClassExtender, con una propiedad denominada `someString`, como se indica a continuación:

```
class ClassExtender extends ClassBase
{
    var someString:String;
}
```

Se pueden usar ambas clases para crear una instancia de clase que se declara con el tipo de datos de ClassBase, pero se crea con el constructor de ClassExtender. Una conversión hacia arriba se considera una operación segura porque la clase base no contiene ninguna propiedad o método que no esté en la subclase.

```
var myClass:ClassBase = new ClassExtender();
```

No obstante, una subclase contiene propiedades o métodos que la clase base no contiene. Por ejemplo, la clase ClassExtender contiene la propiedad `someString`, que no existe en la clase ClassBase. En el modo estándar de ActionScript 3.0 se puede hacer referencia a esta propiedad mediante la instancia de `myClass` sin generar un error de tiempo de compilación, como se muestra en el siguiente ejemplo:

```
var myClass:ClassBase = new ClassExtender();
myClass.someString = "hello";
// no error in ActionScript 3.0 standard mode
```

Operador is

El operador `is`, una de las novedades de ActionScript 3.0, permite comprobar si una variable o expresión es un miembro de un tipo de datos determinado. En versiones anteriores de ActionScript el operador `instanceof` proporcionaba esta funcionalidad, pero en ActionScript 3.0 no se debe utilizar el operador `instanceof` para comprobar la pertenencia a un tipo de datos. Para la verificación manual de tipos se debe utilizar el operador `is` en lugar del operador `instanceof`, ya que la expresión `x instanceof y` simplemente comprueba en la cadena de prototipos de `x` si existe `y` (y en ActionScript 3.0, la cadena de prototipos no proporciona una imagen completa de la jerarquía de herencia).

El operador `is` examina la jerarquía de herencia adecuada y se puede utilizar no sólo para verificar si un objeto es una instancia de una clase específica, sino también en el caso de que un objeto sea una instancia de una clase que implementa una interfaz determinada. En el siguiente ejemplo se crea una instancia de la clase Sprite denominada `mySprite` y se utiliza el operador `is` para comprobar si `mySprite` es una instancia de las clases Sprite y DisplayObject, y si implementa la interfaz IEventDispatcher.

```
var mySprite:Sprite = new Sprite();
trace(mySprite is Sprite); // true
trace(mySprite is DisplayObject);// true
trace(mySprite is IEventDispatcher); // true
```

El operador `is` comprueba la jerarquía de herencia y notifica que `mySprite` es compatible con las clases Sprite y DisplayObject (la clase Sprite es una subclase de la clase DisplayObject). El operador `is` también comprueba si `mySprite` hereda de alguna clase que implementa la interfaz IEventDispatcher. Como la clase Sprite hereda de la clase EventDispatcher, que implementa la interfaz IEventDispatcher, el operador `is` notifica correctamente que `mySprite` implementa la misma interfaz.

En el siguiente ejemplo se muestran las mismas pruebas del ejemplo anterior, pero con `instanceof` en lugar del operador `is`. El operador `instanceof` identifica correctamente que `mySprite` es una instancia de Sprite o DisplayObject, pero devuelve `false` cuando se usa para comprobar si `mySprite` implementa la interfaz IEventDispatcher.

```
trace(mySprite instanceof Sprite); // true
trace(mySprite instanceof DisplayObject);// true
trace(mySprite instanceof IEventDispatcher); // false
```

Operador as

El operador `as`, una de las novedades de ActionScript 3.0, también permite comprobar si una expresión es un miembro de un tipo de datos determinado. Sin embargo, a diferencia del operador `is`, el operador `as` no devuelve un valor booleano. El operador `as` devuelve el valor de la expresión en lugar de `true` y `null` en lugar de `false`. En el siguiente ejemplo se muestran los resultados de utilizar el operador `as` en lugar del operador `is` en el caso sencillo de comprobar si una instancia de Sprite es un miembro de los tipos de datos DisplayObject, IEventDispatcher y Number.

```
var mySprite:Sprite = new Sprite();
trace(mySprite as Sprite); // [object Sprite]
trace(mySprite as DisplayObject); // [object Sprite]
trace(mySprite as IEventDispatcher); // [object Sprite]
trace(mySprite as Number);          // null
```

Al utilizar el operador `as`, el operando de la derecha debe ser un tipo de datos. Si se intenta utilizar una expresión que no sea un tipo de datos como operando de la derecha se producirá un error.

Clases dinámicas

Una clase *dinámica* define un objeto que se puede modificar en tiempo de ejecución añadiendo o modificando propiedades y métodos. Una clase que no es dinámica, como la clase String, es una clase *cerrada*. No es posible añadir propiedades o métodos a una clase cerrada en tiempo de ejecución.

Para crear clases dinámicas se utiliza el atributo `dynamic` al declarar una clase. Por ejemplo, el código siguiente crea una clase dinámica denominada `Protean`:

```
dynamic class Protean
{
    private var privateGreeting:String = "hi";
    public var publicGreeting:String = "hello";
    function Protean()
    {
        trace("Protean instance created");
    }
}
```

Si posteriormente se crea una instancia de la clase `Protean`, se pueden añadir propiedades o métodos fuera de la definición de clase. Por ejemplo, el código siguiente crea una instancia de la clase `Protean` y añade una propiedad denominada `aString` y otra propiedad denominada `aNumber` a la instancia:

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
trace(myProtean.aString, myProtean.aNumber); // testing 3
```

Las propiedades que se añaden a una instancia de una clase dinámica son entidades de tiempo de ejecución, por lo que no se realiza ninguna verificación de tipos en tiempo de ejecución. No se puede añadir una anotación de tipo a una propiedad añadida de esta manera.

También se puede añadir un método a la instancia de `myProtean` definiendo una función y asociando la función a una propiedad de la instancia de `myProtean`. El código siguiente mueve la sentencia trace a un método denominado `traceProtean()`:

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
myProtean.traceProtean = function ()
{
    trace(this.aString, this.aNumber);
};
myProtean.traceProtean(); // testing 3
```

Sin embargo, los métodos creados de esta manera no tienen acceso a ninguna propiedad o método privado de la clase Protean. Además, incluso las referencias a las propiedades o métodos públicos de la clase `Protean` deben calificarse con la palabra clave `this` o el nombre de la clase. En el siguiente ejemplo se muestra el intento de acceso del método `traceProtean()` a las variables privadas y las variables públicas de la clase `Protean`.

```
myProtean.traceProtean = function ()
{
    trace(myProtean.privateGreeting); // undefined
    trace(myProtean.publicGreeting); // hello
};
myProtean.traceProtean();
```

Descripción de los tipos de datos

Los tipos de datos simples son Boolean, int, Null, Number, String, uint y void. Las clases principales de ActionScript también definen los tipos de datos complejos siguientes: Object, Array, Date, Error, Function, RegExp, XML y XMLList.

Tipo de datos Boolean

El tipo de datos Boolean consta de dos valores: `true` y `false`. Ningún otro valor es válido para variables de tipo Boolean. El valor predeterminado de una variable booleana declarada pero no inicializada es `false`.

Tipo de datos int

El tipo de datos int se almacena internamente como un entero de 32 bits y consta del conjunto de enteros entre

-2.147.483.648 (-2^{31}) a 2.147.483,647 ($2^{31} - 1$), ambos incluidos. Las versiones anteriores de ActionScript sólo ofrecían el tipo de datos Number, que se usaba tanto para enteros como para números de coma flotante. En ActionScript 3.0 se tiene acceso a tipos de bajo nivel para enteros de 32 bits con o sin signo. Si la variable no va a usar números de coma flotante, es más rápido y eficaz utilizar el tipo de datos int en lugar del tipo de datos Number.

Para valores enteros que estén fuera del rango de los valores enteros mínimo y máximo, hay que utilizar el tipo de datos Number, que admite valores entre los valores positivo y negativo de 9.007.199.254.740.992 (valores enteros de 53 bits). El valor predeterminado para variables con tipo de datos int es 0.

Tipo de datos Null

El tipo de datos Null (nulo) tiene un único valor: `null`. Éste es el valor predeterminado para el tipo de datos String y para todas las clases que definen tipos de datos complejos, incluida la clase Object. Ninguno de los demás tipos de datos simples, como Boolean, Number, int y uint, contienen el valor `null`. Flash Player y Adobe AIR convertirán el valor `null` en el valor predeterminado adecuado si intenta asignar `null` a variables de tipo Boolean, Number, int o uint. Este tipo de datos no se puede utilizar como una anotación de tipo.

Tipo de datos Number

En ActionScript 3.0, el tipo de datos Number representa enteros, enteros sin signo y números de coma flotante. Sin embargo, para maximizar el rendimiento se recomienda utilizar el tipo de datos Number únicamente para valores enteros que ocupen más que los 32 bits que pueden almacenar los tipos de datos `int` y `uint` o para números de coma flotante. Para almacenar un número de coma flotante se debe incluir una coma decimal en el número. Si se omite un separador decimal, el número se almacenará como un entero.

El tipo de datos Number utiliza el formato de doble precisión de 64 bits especificado en la norma IEEE para aritmética binaria de coma flotante (IEEE-754). Esta norma especifica cómo se almacenan los números de coma flotante utilizando los 64 bits disponibles. Se utiliza un bit para designar si el número es positivo o negativo. El exponente, que se almacena como un número de base 2, utiliza once bits. Los 52 bits restantes se utilizan para almacenar la *cifra significativa* (también denominada *mantisa*), que es el número que se eleva a la potencia indicada por el exponente.

Al utilizar parte de los bits para almacenar un exponente, el tipo de datos Number puede almacenar números de coma flotante considerablemente más grandes que si se utilizaran todos los bits para la mantisa. Por ejemplo, si el tipo de datos Number utilizara los 64 bits para almacenar la mantisa, podría almacenar números hasta 2^{65} -1. Al utilizar 11 bits para almacenar un exponente, el tipo de datos Number puede elevar la mantisa a una potencia de 2^{1023}.

Los valores máximo y mínimo que el tipo Number puede representar se almacenan en propiedades estáticas de la clase Number denominadas `Number.MAX_VALUE` y `Number.MIN_VALUE`.

```
Number.MAX_VALUE == 1.79769313486231e+308
Number.MIN_VALUE == 4.940656458412467e-324
```

Aunque este rango de números es enorme, se pierde en precisión. El tipo de datos Number utiliza 52 bits para almacenar la mantisa y, como consecuencia, los números que requieren más de 52 bits para una representación precisa, como la fracción 1/3, son sólo aproximaciones. Si la aplicación requiere precisión absoluta con números decimales, hay que utilizar software que implemente aritmética decimal de coma flotante en lugar de aritmética binaria de coma flotante.

Al almacenar valores enteros con el tipo de datos Number, sólo se utilizarán los 52 bits de la mantisa. El tipo de datos Number utiliza estos 52 bits y un bit oculto especial para representar enteros entre -9.007.199.254.740.992 (-2^{53}) y 9.007.199.254.740.992 (2^{53}).

Flash Player y Adobe AIR utilizan el valor `NaN` no sólo como el valor predeterminado para las variables de tipo `Number`, sino también como resultado de cualquier operación que debiera devolver un número y no lo haga. Por ejemplo, si se intenta calcular la raíz cuadrada de un número negativo, el resultado será `NaN`. Otros valores especiales de Number son *infinito positivo* e *infinito negativo* .

Nota: *el resultado de la división por `0` sólo es `NaN` si el divisor también es `0`. La división por `0` produce `infinity` cuando el dividendo es positivo o `-infinity` cuando el dividendo es negativo.*

Tipo de datos String

El tipo de datos String representa una secuencia de caracteres de 16 bits. Las cadenas se almacenan internamente como caracteres Unicode empleando el formato UTF-16. Las cadenas son valores inmutables, igual que en el lenguaje de programación Java. Una operación sobre un valor de cadena (String) devuelve una nueva instancia de la cadena. El valor predeterminado de una variable declarada con el tipo de datos String es `null`. El valor `null` no es lo mismo que la cadena vacía (`""`), aunque ambos representan la ausencia de caracteres.

Tipo de datos uint

El tipo de datos uint se almacena internamente como un entero sin signo de 32 bits y consta del conjunto de enteros entre 0 y 4.294.967.295 (2^{32}- 1), ambos incluidos. El tipo de datos uint debe utilizarse en circunstancias especiales que requieran enteros no negativos. Por ejemplo, se debe utilizar el tipo de datos uint para representar valores de colores de píxeles, ya que el tipo de datos int tiene un bit de signo interno que no es apropiado para procesar valores de colores. Para valores enteros más grandes que el valor uint máximo, se debe utilizar el tipo de datos Number, que puede procesar valores enteros de 53 bits. El valor predeterminado para variables con tipo de datos uint es 0.

Tipo de datos Void

El tipo de datos void tiene un único valor: `undefined`. En las versiones anteriores de ActionScript, `undefined` era el valor predeterminado de las instancias de la clase Object. En ActionScript 3.0, el valor predeterminado de las instancias de Object es `null`. Si se intenta asignar el valor `undefined` a una instancia de la clase Object, Flash Player o Adobe AIR convertirán el valor en `null`. Sólo se puede asignar un valor `undefined` a variables que no tienen tipo. Las variables sin tipo son variables que no tienen anotación de tipo o utilizan el símbolo de asterisco (`*`) como anotación de tipo. Sólo se puede usar `void` como anotación de tipo devuelto.

Tipo de datos Object

El tipo de datos Object (objeto) se define mediante la clase Object. La clase Object constituye la clase base para todas las definiciones de clase en ActionScript. La versión del tipo de datos Object en ActionScript 3.0 difiere de la de versiones anteriores en tres aspectos. En primer lugar, el tipo de datos Object ya no es el tipo de datos predeterminado que se asigna a las variables sin anotación de tipo. En segundo lugar, el tipo de datos Object ya no incluye el valor `undefined` que se utilizaba como valor predeterminado de las instancias de Object. Por último, en ActionScript 3.0, el valor predeterminado de las instancias de la clase Object es `null`.

En versiones anteriores de ActionScript, a una variable sin anotación de tipo se le asignaba automáticamente el tipo de datos Object. Esto ya no es así en ActionScript 3.0, que incluye el concepto de variable sin tipo. Ahora se considera que las variables sin anotación de tipo no tienen tipo. Si se prefiere dejar claro a los lectores del código que la intención es dejar una variable sin tipo, se puede utilizar el nuevo símbolo de asterisco (`*`) para la anotación de tipo, que equivale a omitir una anotación de tipo. En el siguiente ejemplo se muestran dos sentencias equivalentes que declaran una variable sin tipo `x`:

```
var x
var x:*
```

Sólo las variables sin tipo pueden contener el valor `undefined`. Si se intenta asignar el valor `undefined` a una variable que tiene un tipo de datos, Flash Player o Adobe AIR convertirán el valor `undefined` en el valor predeterminado de dicho tipo de datos. En el caso de las instancias del tipo de datos Object, el valor predeterminado es `null`, según el cual Flash Player o Adobe AIR convertirán el valor `undefined` en `null` si intenta asignar `undefined` a una instancia Object.

Conversiones de tipos

Se dice que se produce una conversión de tipo cuando se transforma un valor en otro valor con un tipo de datos distinto. Las conversiones de tipo pueden ser *implícitas* o *explícitas*. La conversión implícita, también denominada *coerción*, en ocasiones es realizada en tiempo de ejecución por Flash Player o Adobe AIR. Por ejemplo, si a una variable del tipo de datos Boolean se le asigna el valor 2, Flash Player o Adobe AIR convertirán este valor en el valor booleano `true` antes de asignar el valor a la variable. La conversión explícita, también denominada *conversión*, se produce cuando el código ordena al compilador que trate una variable de un tipo de datos como si perteneciera a un tipo de datos distinto. Si se usan valores simples, la conversión convierte realmente los valores de un tipo de datos a otro. Para convertir un objeto a otro tipo, hay que incluir el nombre del objeto entre paréntesis y anteponerle el nombre del nuevo tipo. Por ejemplo, el siguiente código toma un valor booleano y lo convierte en un entero:

```
var myBoolean:Boolean = true;
var myINT:int = int(myBoolean);
trace(myINT); // 1
```

Conversiones implícitas

Las conversiones implícitas se realizan en tiempo de ejecución en algunos contextos:

- En sentencias de asignación
- Cuando se pasan valores como argumentos de función
- Cuando se devuelven valores desde funciones
- En expresiones que utilizan determinados operadores, como el operador suma (+)

  Para tipos definidos por el usuario, las conversiones implícitas se realizan correctamente cuando el valor que se va a convertir es una instancia de la clase de destino o una clase derivada de la clase de destino. Si una conversión implícita no se realiza correctamente, se producirá un error. Por ejemplo, el código siguiente contiene una conversión implícita correcta y otra incorrecta:

  ```
  class A {}
  class B extends A {}

  var objA:A = new A();
  var objB:B = new B();
  var arr:Array = new Array();

  objA = objB; // Conversion succeeds.
  objB = arr; // Conversion fails.
  ```

  Para tipos simples, las conversiones implícitas se realizan llamando a los mismos algoritmos internos de conversión que utilizan las funciones de conversión explícita. En las secciones siguientes se describen en mayor detalle estas conversiones de tipos simples.

Conversiones explícitas

Resulta útil usar conversiones explícitas cuando se compila en modo estricto, ya que a veces no se desea que una discordancia de tipos genere un error en tiempo de compilación. Esto puede ocurrir, por ejemplo, cuando se sabe que la coerción convertirá los valores correctamente en tiempo de ejecución. Por ejemplo, al trabajar con datos recibidos desde un formulario, puede ser interesante basarse en la coerción para convertir determinados valores de cadena en valores numéricos. El código siguiente genera un error de tiempo de compilación aunque se ejecuta correctamente en modo estándar:

```
var quantityField:String = "3";
var quantity:int = quantityField; // compile time error in strict mode
```

Si se desea seguir utilizando el modo estricto pero se quiere convertir la cadena en un entero, se puede utilizar la conversión explícita de la manera siguiente:

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // Explicit conversion succeeds.
```

Conversión a int, uint y Number

Cualquier tipo de datos se puede convertir en uno de los tres tipos numéricos siguientes: int, uint y Number. Si por algún motivo Flash Player o Adobe AIR no pueden convertir el número, se asignará el valor predeterminado 0 a los tipos de datos int y uint, y el valor predeterminado de `NaN` se asignará al tipo de datos Number. Si se convierte un valor booleano a un número, `true` se convierte en el valor 1 y `false` se convierte en el valor 0.

```
var myBoolean:Boolean = true;
var myUINT:uint = uint(myBoolean);
var myINT:int = int(myBoolean);
var myNum:Number = Number(myBoolean);
trace(myUINT, myINT, myNum); // 1 1 1
myBoolean = false;
myUINT = uint(myBoolean);
myINT = int(myBoolean);
myNum = Number(myBoolean);
trace(myUINT, myINT, myNum); // 0 0 0
```

Los valores de cadena que sólo contienen dígitos pueden convertirse correctamente en uno de los tipos numéricos. Los tipos numéricos también pueden convertir cadenas que parecen números negativos o cadenas que representan un valor hexadecimal (por ejemplo, `0x1A`). El proceso de conversión omite los caracteres de espacio en blanco iniciales y finales del valor de cadena. También se puede convertir cadenas que parecen números de coma flotante mediante `Number()`. La inclusión de un separador decimal hace que `uint()` e `int()` devuelvan un entero, truncando el separador decimal y los caracteres que siguen. Por ejemplo, los siguientes valores de cadena pueden convertirse en números:

```
trace(uint("5")); // 5
trace(uint("-5")); // 4294967291. It wraps around from MAX_VALUE
trace(uint(" 27 ")); // 27
trace(uint("3.7")); // 3
trace(int("3.7")); // 3
trace(int("0x1A")); // 26
trace(Number("3.7")); // 3.7
```

Los valores de cadena que contienen caracteres no numéricos devuelven 0 cuando se convierten con `int()` o `uint()`, y `NaN` cuando se convierten con`Number()`. El proceso de conversión omite el espacio en blanco inicial y final, pero devuelve 0 o `NaN` si una cadena contiene espacio en blanco separando dos números.

```
trace(uint("5a")); // 0
trace(uint("ten")); // 0
trace(uint("17 63")); // 0
```

En ActionScript 3.0 la función `Number()` ya no admite números octales (de base 8). Si se suministra una cadena con un cero inicial a la función `Number()` de ActionScript 2.0, el número se interpreta como un número octal y se convierte en su equivalente decimal. Esto no es así con la función `Number()` de ActionScript 3.0, que omite el cero inicial. Por ejemplo, el código siguiente genera resultados distintos cuando se compila con versiones distintas de ActionScript:

```
trace(Number("044"));
// ActionScript 3.0 44
// ActionScript 2.0 36
```

La conversión no es necesaria cuando se asigna un valor de un tipo numérico a una variable de un tipo numérico distinto. Incluso en modo estricto, los tipos numéricos se convierten implícitamente a los otros tipos numéricos. Esto significa que en algunos casos pueden producirse valores inesperados cuando se supera el rango de un tipo. Todos los ejemplos siguientes se compilan en modo estricto, aunque algunos generarán valores inesperados:

```
var myUInt:uint = -3; // Assign int/Number value to uint variable
trace(myUInt); // 4294967293

var myNum:Number = sampleUINT; // Assign int/uint value to Number variable
trace(myNum) // 4294967293

var myInt:int = uint.MAX_VALUE + 1; // Assign Number value to uint variable
trace(myInt); // 0

myInt = int.MAX_VALUE + 1; // Assign uint/Number value to int variable
trace(myInt); // -2147483648
```

En la tabla siguiente se resumen los resultados de convertir a los tipos de datos Number, int o uint desde otros tipos de datos.

Tipo de datos o valor	Resultado de la conversión a Number, int o uint
Boolean	Si el valor es `true`, 1; de lo contrario, 0.
Date	Representación interna del objeto Date, que es el número de milisegundos transcurridos desde la medianoche del 1 de enero de 1970, hora universal.
`null`	0
Object	Si la instancia es `null` y se convierte a Number, `NaN`; de lo contrario, 0.
String	Un número si Flash Player o Adobe AIR pueden convertir la cadena en un número; de lo contrario, `NaN` si se convierte en Number, o 0 si se convierte a int o uint.
`undefined`	Si se convierte a Number, `NaN`; si se convierte a int o uint, 0.

Conversión a Boolean

La conversión a Boolean desde cualquiera de los tipos de datos numéricos (uint, int y Number) produce `false` si el valor numérico es 0 y `true` en caso contrario. Para el tipo de datos Number, el valor `NaN` también produce `false`. En el siguiente ejemplo se muestran los resultados de convertir los números -1, 0 y 1:

```
var myNum:Number;
for (myNum = -1; myNum<2; myNum++)
{
    trace("Boolean(" + myNum +") is " + Boolean(myNum));
}
```

El resultado del ejemplo muestra que de los tres números, sólo 0 devuelve un valor `false`:

```
Boolean(-1) is true
Boolean(0) is false
Boolean(1) is true
```

La conversión a Boolean desde un valor String devuelve `false` si la cadena es `null` o una cadena vacía (`""`). De lo contrario, devuelve `true`.

```
var str1:String; // Uninitialized string is null.
trace(Boolean(str1)); // false

var str2:String = ""; // empty string
trace(Boolean(str2)); // false

var str3:String = " "; // white space only
trace(Boolean(str3)); // true
```

La conversión a Boolean desde una instancia de la clase Object devuelve `false` si la instancia es `null` y `true` en caso contrario:

```
var myObj:Object; // Uninitialized object is null.
trace(Boolean(myObj)); // false

myObj = new Object(); // instantiate
trace(Boolean(myObj)); // true
```

En modo estricto, las variables Boolean reciben un tratamiento especial en el sentido de que se puede asignar valores de cualquier tipo de datos a una variable Boolean sin realizar una conversión. La coerción implícita desde todos los tipos de datos al tipo de datos Boolean se produce incluso en modo estricto. Es decir, a diferencia de lo que ocurre para casi todos los demás tipos de datos, la conversión a Boolean no es necesaria para evitar errores en modo estricto. Todos los ejemplos siguientes se compilan en modo estricto y se comportan de la manera esperada en tiempo de ejecución:

```
var myObj:Object = new Object(); // instantiate
var bool:Boolean = myObj;
trace(bool); // true
bool = "random string";
trace(bool); // true
bool = new Array();
trace(bool); // true
bool = NaN;
trace(bool); // false
```

En la tabla siguiente se resumen los resultados de convertir al tipo de datos Boolean desde otros tipos de datos:

Tipo de datos o valor	Resultado de la conversión a Boolean
String	`false` si el valor es `null` o la cadena vacía (`""`); `true` en caso contrario.
`null`	`false`
Number, int o uint	`false` si el valor es `NaN` o 0; `true` en caso contrario.
Object	`false` si la instancia es `null`; `true` en caso contrario.

Conversión a String

La conversión al tipo de datos String desde cualquiera de los tipos de datos numéricos devuelve una representación del número como una cadena. La conversión al tipo de datos String desde un valor booleano devuelve la cadena `"true"` si el valor es `true` y devuelve la cadena `"false"` si el valor es `false`.

La conversión al tipo de datos String desde una instancia de la clase Object devuelve la cadena `"null"` si la instancia es `null`. De lo contrario, la conversión al tipo String de la clase Object devuelve la cadena `"[object Object]"`.

La conversión a String desde una instancia de la clase Array devuelve una cadena que consta de una lista delimitada por comas de todos los elementos del conjunto. Por ejemplo, la siguiente conversión al tipo de datos String devuelve una cadena que contiene los tres elementos del conjunto:

```
var myArray:Array = ["primary", "secondary", "tertiary"];
trace(String(myArray)); // primary,secondary,tertiary
```

La conversión a String desde una instancia de la clase Date devuelve una representación de cadena de la fecha que contiene la instancia. Por ejemplo, el ejemplo siguiente devuelve una representación de cadena de la instancia de la clase Date (la salida muestra el resultado para el horario de verano de la costa del Pacífico de EE.UU.):

```
var myDate:Date = new Date(2005,6,1);
trace(String(myDate)); // Fri Jul 1 00:00:00 GMT-0700 2005
```

En la tabla siguiente se resumen los resultados de convertir al tipo de datos String desde otros tipos de datos.

Tipo de datos o valor	Resultado de la conversión a String
Array	Una cadena que consta de todos los elementos de conjunto.
Boolean	`"true"` o `"false"`
Date	Una representación de cadena del objeto Date.
`null`	`"null"`
Number, int o uint	Una representación de cadena del número.
Object	Si la instancia es null, `"null"`; de lo contrario, `"[object Object]"`.

Sintaxis

La sintaxis de un lenguaje define un conjunto de reglas que deben cumplirse al escribir código ejecutable.

Distinción entre mayúsculas y minúsculas

El lenguaje ActionScript 3.0 distingue mayúsculas de minúsculas. Los identificadores que sólo se diferencien en mayúsculas o minúsculas se considerarán identificadores distintos. Por ejemplo, el código siguiente crea dos variables distintas:

```
var num1:int;
var Num1:int;
```

Sintaxis con punto

El operador de punto (.) permite acceder a las propiedades y los métodos de un objeto. La sintaxis con punto permite hacer referencia a una propiedad o un método de clase mediante un nombre de instancia, seguido del operador punto y el nombre de la propiedad o el método. Por ejemplo, considere la siguiente definición de clase:

```
class DotExample
{
    public var prop1:String;
    public function method1():void {}
}
```

La sintaxis con punto permite acceder a la propiedad `prop1` y al método `method1()` utilizando el nombre de la instancia creada en el código siguiente:

```
var myDotEx:DotExample = new DotExample();
myDotEx.prop1 = "hello";
myDotEx.method1();
```

Se puede utilizar la sintaxis con punto al definir paquetes. El operador punto se utiliza para hacer referencia a paquetes anidados. Por ejemplo, la clase EventDispatcher reside en un paquete denominado events que está anidado dentro del paquete denominado flash. Se puede hacer referencia al paquete events mediante la siguiente expresión:

```
flash.events
```

También se puede hacer referencia a la clase EventDispatcher mediante esta expresión:

```
flash.events.EventDispatcher
```

Sintaxis con barras diagonales

ActionScript 3.0 no admite la sintaxis con barras diagonales. Esta sintaxis se utilizaba en versiones anteriores de ActionScript para indicar la ruta a un clip de película o una variable.

Literales

Un *literal* es un valor que aparece directamente en el código. Todos los ejemplos siguientes son literales:

```
17
"hello"
-3
9.4
null
undefined
true
false
```

Los literales también pueden agruparse para formar literales compuestos. Los literales de conjunto se escriben entre corchetes (`[]`) y utilizan la coma para separar los elementos de conjunto.

Un literal de conjunto puede utilizarse para inicializar un conjunto. En los siguientes ejemplos se muestran dos conjuntos que se inicializan mediante literales de conjunto. Se puede utilizar la sentencia `new` y pasar el literal compuesto como parámetro al constructor de la clase Array, pero también se pueden asignar valores literales directamente al crear instancias de las siguientes clases principales de ActionScript: Object, Array, String, Number, int, uint, XML, XMLList y Boolean.

```
// Use new statement.
var myStrings:Array = new Array(["alpha", "beta", "gamma"]);
var myNums:Array = new Array([1,2,3,5,8]);

// Assign literal directly.
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1,2,3,5,8];
```

Los literales también se pueden utilizar para inicializar un objeto genérico. Un objeto genérico es una instancia de la clase Object. Los literales de objetos se escriben entre llaves (`{}`) y utilizan la coma para separar las propiedades del objeto. Cada propiedad se declara mediante el signo de dos puntos (`:`), que separa el nombre de la propiedad del valor de la propiedad.

Se puede crear un objeto genérico utilizando la sentencia `new` y pasar el literal de objeto como parámetro al constructor de la clase Object, o bien asignar el literal de objeto directamente a la instancia que se está declarando. En el siguiente ejemplo se muestran dos formas de crear un nuevo objeto genérico y se inicializa el objeto con tres propiedades (`propA`, `propB` y `propC`) establecidas en los valores 1, 2 y 3 respectivamente:

```
// Use new statement and add properties.
var myObject:Object = new Object();
myObject.propA = 1;
myObject.propB = 2;
myObject.propC = 3;

// Assign literal directly.
var myObject:Object = {propA:1, propB:2, propC:3};
```

Para obtener más información, consulte "Fundamentos de la utilización de cadenas" en la página 144, "Fundamentos de la utilización de expresiones regulares" en la página 211 e "Inicialización de variables XML" en la página 240.

Signos de punto y coma

Se puede utilizar el signo de punto y coma (`;`) para finalizar una sentencia. Como alternativa, si se omite el signo de punto y coma, el compilador dará por hecho que cada línea de código representa a una sentencia independiente. Como muchos programadores están acostumbrados a utilizar el signo de punto y coma para indicar el final de una sentencia, el código puede ser más legible si se usan siempre signos de punto y coma para finalizar las sentencias.

El uso del punto y coma para terminar una sentencia permite colocar más de una sentencia en una misma línea, pero esto hará que el código resulte más difícil de leer.

Paréntesis

Los paréntesis (`()`) se pueden utilizar de tres modos diferentes en ActionScript 3.0. En primer lugar, se pueden utilizar para cambiar el orden de las operaciones de una expresión. Las operaciones agrupadas entre paréntesis siempre se ejecutan primero. Por ejemplo, se utilizan paréntesis para modificar el orden de las operaciones en el código siguiente:

```
trace(2 + 3 * 4); // 14
trace((2 + 3) * 4); // 20
```

En segundo lugar, se pueden utilizar paréntesis con el operador coma (`,`) para evaluar una serie de expresiones y devolver el resultado de la expresión final, como se indica en el siguiente ejemplo:

```
var a:int = 2;
var b:int = 3;
trace((a++, b++, a+b)); // 7
```

Por último, se pueden utilizar paréntesis para pasar uno o más parámetros a funciones o métodos, como se indica en el siguiente ejemplo, que pasa un valor String a la función `trace()`:

```
trace("hello"); // hello
```

Comentarios

El código de ActionScript 3.0 admite dos tipos de comentarios: comentarios de una sola línea y comentarios multilínea. Estos mecanismos para escribir comentarios son similares a los equivalentes de C++ y Java. El compilador omitirá el texto marcado como un comentario.

Los comentarios de una sola línea empiezan por dos caracteres de barra diagonal (`//`) y continúan hasta el final de la línea. Por ejemplo, el código siguiente contiene un comentario de una sola línea:

```
var someNumber:Number = 3; // a single line comment
```

Los comentarios multilínea empiezan con una barra diagonal y un asterisco (`/*`) y terminan con un asterisco y una barra diagonal (`*/`).

```
/* This is multiline comment that can span
more than one line of code. */
```

Palabras clave y palabras reservadas

Las palabras reservadas son aquellas que no se pueden utilizar como identificadores en el código porque su uso está reservado para ActionScript. Incluyen las *palabras clave léxicas*, que son eliminadas del espacio de nombres del programa por el compilador. El compilador notificará un error si se utiliza una palabra clave léxica como un identificador. En la tabla siguiente se muestran las palabras clave léxicas de ActionScript 3.0.

as	break	case	catch
clase	const	continue	default
delete	do	else	extends
false	finally	for	function
if	implements	import	in
instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof
use	var	void	while
with			

Hay un pequeño conjunto de palabras clave, denominadas *palabras clave sintácticas*, que se pueden utilizar como identificadores, pero que tienen un significado especial en determinados contextos. En la tabla siguiente se muestran las palabras clave sintácticas de ActionScript 3.0.

each	get	set	namespace
include	dynamic	final	native
override	static		

También hay varios identificadores que a veces se llaman *futuras palabras reservadas*. ActionScript 3.0 no reserva estos identificadores, aunque el software que incorpore ActionScript 3.0 podrá tratar algunos de ellos como palabras clave. Es posible que pueda utilizar un gran número de estos identificadores en su código, pero Adobe recomienda que no se utilicen ya que pueden aparecer como palabras clave en una versión posterior de dicho lenguaje.

abstract	boolean	byte	cast
char	debugger	double	enum
export	float	goto	intrinsic
long	prototype	short	synchronized
throws	to	transient	type
virtual	volatile		

Constantes

ActionScript 3.0 admite la sentencia `const`, que se puede utilizar para crear constantes. Las constantes son propiedades con un valor fijo que no se puede modificar. Se puede asignar un valor a una constante una sola vez y la asignación debe realizarse cerca de la declaración de la constante. Por ejemplo, si se declara una constante como un miembro de una clase, se puede asignar un valor a esa constante únicamente como parte de la declaración o dentro del constructor de la clase.

El código siguiente declara dos constantes. Se asigna un valor a la primera constante, `MINIMUM`, como parte de la sentencia de declaración. A la segunda constante, `MAXIMUM`, se le asigna un valor en el constructor. Es necesario tener en cuenta que este ejemplo sólo compila en modo estándar, ya que el modo estricto sólo permite asignar un valor de constante en tiempo de inicialización.

```
class A
{
    public const MINIMUM:int = 0;
    public const MAXIMUM:int;

    public function A()
    {
        MAXIMUM = 10;
    }
}

var a:A = new A();
trace(a.MINIMUM); // 0
trace(a.MAXIMUM); // 10
```

Se producirá un error si se intenta asignar de otra manera un valor inicial a una constante. Por ejemplo, si se intenta establecer el valor inicial de `MAXIMUM` fuera de la clase, se producirá un error en tiempo de ejecución.

```
class A
{
public const MINIMUM:int = 0;
public const MAXIMUM:int;
}

var a:A = new A();
a["MAXIMUM"] = 10; // run-time error
```

ActionScript 3.0 define una amplia gama de constantes para su uso. Por convención, en ActionScript las constantes se escriben en mayúsculas y las palabras que las forman se separan mediante el carácter de subrayado (_). Por ejemplo, la definición de la clase MouseEvent utiliza esta convención de nomenclatura para sus constantes, cada una de las cuales representa un evento relacionado con una entrada del ratón:

```
package flash.events
{
    public class MouseEvent extends Event
    {
    public static const CLICK:String = "click";
    public static const DOUBLE_CLICK:String = "doubleClick";
    public static const MOUSE_DOWN:String = "mouseDown";
    public static const MOUSE_MOVE:String = "mouseMove";
    ...
    }
}
```

Operadores

Los operadores son funciones especiales que se aplican a uno o más operandos y devuelven un valor. Un *operando* es un valor (generalmente un literal, una variable o una expresión) que se usa como entrada de un operador. Por ejemplo, en el código siguiente, los operadores de suma (+) y multiplicación (*) se usan con tres operandos literales (2, 3 y 4) para devolver un valor. A continuación, el operador de asignación (=) usa este valor para asignar el valor devuelto, 14, a la variable `sumNumber`.

```
var sumNumber:uint = 2 + 3 * 4; // uint = 14
```

Los operadores pueden ser unarios, binarios o ternarios. Un operador *unario* se aplica a un operando. Por ejemplo, el operador de incremento (++) es un operador unario porque se aplica a un solo operando. Un operador *binario* se aplica a dos operandos. Por ejemplo, el operador división (/) se aplica a dos operandos. Un operador *ternario* se aplica a tres operandos. Por ejemplo, el operador condicional (?:) se aplica a tres operandos.

Algunos operadores están *sobrecargados*, lo que significa que se comportan de distinta manera en función del tipo o la cantidad de operandos que se les pase. El operador suma (+) es un ejemplo de un operador sobrecargado que se comporta de distinta manera en función del tipo de datos de los operandos. Si ambos operandos son números, el operador suma devuelve la suma de los valores. Si ambos operandos son cadenas, el operador suma devuelve la concatenación de los dos operandos. En el siguiente ejemplo de código se muestra cómo cambia el comportamiento del operador en función de los operandos:

```
trace(5 + 5); // 10
trace("5" + "5"); // 55
```

Los operadores también pueden comportarse de distintas maneras en función del número de operandos suministrados. El operador resta (-) es la vez un operador unario y un operador binario. Si se le suministra un solo operando, el operador resta devuelve como resultado la negación del operando. Si se le suministran dos operandos, el operador resta devuelve la diferencia de los operandos. En el siguiente ejemplo se muestra el operador resta usado primero como un operador unario y después como un operador binario.

```
trace(-3); // -3
trace(7 - 2); // 5
```

Precedencia y asociatividad de operadores

La precedencia y asociatividad de los operadores determina el orden en que se procesan los operadores. Aunque para aquellos usuarios familiarizados con la programación aritmética puede parecer natural que el compilador procese el operador de multiplicación (*) antes que el operador de suma (+), el compilador necesita instrucciones explícitas sobre qué operadores debe procesar primero. Dichas instrucciones se conocen colectivamente como *precedencia de operadores*. ActionScript establece una precedencia de operadores predeterminada que se puede modificar utilizando el operador paréntesis (()). Por ejemplo, el código siguiente modifica la precedencia predeterminada del ejemplo anterior para forzar al compilador a procesar el operador suma antes que el operador producto:

```
var sumNumber:uint = (2 + 3) * 4; // uint == 20
```

Pueden darse situaciones en las que dos o más operadores con la misma precedencia aparezcan en la misma expresión. En estos casos, el compilador utiliza las reglas de *asociatividad* para determinar qué operador se procesa primero. Todos los operadores binarios, salvo los operadores de asignación, tienen *asociatividad desde la izquierda*, lo que significa que los operadores de la izquierda se procesan antes que los operadores de la derecha. Los operadores de asignación y el operador condicional (?:) tienen *asociatividad desde la derecha*, lo que significa que los operadores de la derecha se procesan antes que los operadores de la izquierda.

Consideremos, por ejemplo, los operadores menor que (<) y mayor que (>), que tienen la misma precedencia. Si ambos operadores se utilizan en la misma expresión, el operador de la izquierda se procesará en primer lugar porque ambos operadores tienen asociatividad desde la izquierda. Esto significa que las dos sentencias siguientes generan el mismo resultado:

```
trace(3 > 2 < 1); // false
trace((3 > 2) < 1); // false
```

El operador mayor que se procesa primero, lo que devuelve un valor `true`, ya que el operando 3 es mayor que el operando 2. A continuación, se pasa el valor `true` al operador menor que, junto con el operando 1. El código siguiente representa este estado intermedio:

```
trace((true) < 1);
```

El operador menor que convierte el valor `true` en el valor numérico 1 y compara dicho valor numérico con el segundo operando 1 para devolver el valor `false` (el valor 1 no es menor que 1).

```
trace(1 < 1); // false
```

Se puede modificar la asociatividad predeterminada desde la izquierda con el operador paréntesis. Se puede ordenar al compilador que procese primero el operador menor que escribiendo dicho operador y sus operandos entre paréntesis. En el ejemplo siguiente se utiliza el operador paréntesis para producir un resultado diferente utilizando los mismos números que en el ejemplo anterior:

```
trace(3 > (2 < 1)); // true
```

El operador menor que se procesa primero, lo que devuelve un valor `false`, ya que el operando 2 no es menor que el operando 1. A continuación, se pasa el valor `false` al operador mayor que, junto con el operando 3. El código siguiente representa este estado intermedio:

```
trace(3 > (false));
```

El operador mayor que convierte el valor `false` en el valor numérico 0 y compara dicho valor numérico con el otro operando 3 para devolver el valor `true` (el valor 3 es mayor que 0).

```
trace(3 > 0); // true
```

En la tabla siguiente se muestran los operadores de ActionScript 3.0 por orden decreciente de precedencia. Cada fila de la tabla contiene operadores de la misma precedencia. Cada fila de operadores tiene precedencia superior a la fila que aparece debajo de ella en la tabla.

Grupo	**Operadores**
Primario	`[] {x:y} () f(x) new x.y x[y] <></> @ :: ..`
Sufijo	`x++ x--`
Unario	`++x --x + - ~ ! delete typeof void`
Multiplicativo	`* / %`
Aditivo	`+ -`
Desplazamiento en modo bit	`<< >> >>>`
Relacional	`< > <= >= as in instanceof is`
Igualdad	`== != === !==`
AND en modo bit	`&`
XOR en modo bit	`^`

Grupo	Operadores
OR en modo bit	`|`
AND lógico	`&&`
OR lógico	`||`
Condicional	`?:`
Asignación	`= *= /= %= += -= <<= >>= >>>= &= ^= |=`
Coma	`,`

Operadores principales

Los operadores principales incluyen los que se utilizan para crear literales Array y Object, agrupar expresiones, llamar a funciones, crear instancias de clase y acceder a propiedades.

Todos los operadores principales, indicados en la tabla siguiente, tienen la misma precedencia. Los operadores que forman parte de la especificación E4X se indican mediante la notación (E4X).

Operador	Operación realizada
`[]`	Inicializa un conjunto
`{x:y}`	Inicializa un objeto
`()`	Agrupa expresiones
`f(x)`	Llama a una función
`new`	Llama a un constructor
`x.y x[y]`	Accede a una propiedad
`<></>`	Inicializa un objeto XMLList (E4X)
`@`	Accede a un atributo (E4X)
`::`	Califica un nombre (E4X)
`..`	Accede a un elemento XML descendiente (E4X)

Operadores de sufijo

Los operadores de sufijo se aplican a un operador para aumentar o reducir el valor. Aunque estos operadores son unarios, se clasifican por separado del resto de los operadores unarios debido a su mayor precedencia y a su comportamiento especial. Al utilizar un operador de sufijo como parte de una expresión mayor, el valor de la expresión se devuelve antes de que se procese el operador de sufijo. Por ejemplo, el siguiente código muestra cómo se devuelve el valor de la expresión `xNum++` antes de que se incremente el valor:

```
var xNum:Number = 0;
trace(xNum++); // 0
trace(xNum); // 1
```

Todos los operadores de sufijo, indicados en la tabla siguiente, tienen la misma precedencia:

Operador	Operación realizada
`++`	Incremento (sufijo)
`--`	Decremento (sufijo)

Operadores unarios

Los operadores unarios se aplican a un operando. Los operadores de incremento (`++`) y decremento (`--`) de este grupo son *operadores* de *prefijo*, lo que significa que aparecen delante del operando en una expresión. Los operadores de prefijo difieren de los correspondientes operadores de sufijo en que la operación de incremento o decremento se realiza antes de que se devuelva el valor de la expresión global. Por ejemplo, el siguiente código muestra cómo se devuelve el valor de la expresión `++xNum` después de que se incremente el valor:

```
var xNum:Number = 0;
trace(++xNum); // 1
trace(xNum); // 1
```

Todos los operadores unarios, indicados en la tabla siguiente, tienen la misma precedencia:

Operador	Operación realizada
`++`	Incremento (prefijo)
`--`	Decremento (prefijo)
`+`	Unario +
`-`	Unario - (negación)
`!`	NOT lógico
`~`	NOT en modo bit
`delete`	Elimina una propiedad
`typeof`	Devuelve información de tipo
`void`	Devuelve un valor no definido

Operadores multiplicativos

Los operadores multiplicativos toman dos operandos y realizan cálculos de multiplicación, división o módulo.

Todos los operadores multiplicativos, indicados en la tabla siguiente, tienen la misma precedencia:

Operador	Operación realizada
`*`	Multiplicación
`/`	División
`%`	Módulo

Operadores aditivos

Los operadores aditivos se aplican a dos operandos y realizan cálculos de suma y resta. Todos los operadores aditivos, indicados en la tabla siguiente, tienen la misma precedencia:

Operador	Operación realizada
+	Suma
-	Resta

Operadores de desplazamiento en modo bit

Los operadores de desplazamiento en modo bit se aplican a dos operandos y desplazan los bits del primer operando según lo especificado por el segundo operando. Todos los operadores de desplazamiento en modo de bit, indicados en la tabla siguiente, tienen la misma precedencia:

Operador	Operación realizada
<<	Desplazamiento a la izquierda en modo bit
>>	Desplazamiento a la derecha en modo bit
>>>	Desplazamiento a la derecha en modo bit sin signo

Operadores relacionales

Los operadores relacionales se aplican a dos operandos, comparan sus valores y devuelven un valor booleano. Todos los operadores relacionales, indicados en la tabla siguiente, tienen la misma precedencia:

Operador	Operación realizada
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
as	Comprueba el tipo de datos
in	Comprueba las propiedades de objetos
instanceof	Comprueba una cadena de prototipos
is	Comprueba el tipo de datos

Operadores de igualdad

Los operadores de igualdad se aplican a dos operandos, comparan sus valores y devuelven un valor booleano. Todos los operadores de igualdad, indicados en la tabla siguiente, tienen la misma precedencia:

Operador	Operación realizada
==	Igualdad
!=	Desigualdad
===	Igualdad estricta
!==	Desigualdad estricta

Operadores lógicos en modo bit

Los operadores lógicos en modo bit se aplican a dos operandos y realizan operaciones lógicas a nivel de bits. Estos operadores, que tienen una precedencia diferente, se enumeran en la tabla siguiente por orden decreciente de precedencia:

Operador	Operación realizada
`&`	AND en modo bit
`^`	XOR en modo bit
`|`	OR en modo bit

Operadores lógicos

Los operadores lógicos se aplican a dos operandos y devuelven un resultado booleano. Estos operadores, que tienen distintas precedencias, se enumeran en la tabla siguiente por orden decreciente de precedencia:

Operador	Operación realizada
`&&`	AND lógico
`||`	OR lógico

Operador condicional

El operador condicional es un operador ternario, lo que significa que se aplica a tres operandos. El operador condicional es un método abreviado para aplicar la sentencia condicional `if..else`.

Operador	Operación realizada
`?:`	Condicional

Operadores de asignación

Los operadores de asignación se aplican a dos operandos y asignan un valor a un operando en función del valor del otro operando. Todos los operadores de asignación, indicados en la tabla siguiente, tienen la misma precedencia:

Operador	Operación realizada
`=`	Asignación
`*=`	Asignación de multiplicación
`/=`	Asignación de división
`%=`	Asignación de módulo
`+=`	Asignación de suma
`-=`	Asignación de resta
`<<=`	Asignación de desplazamiento a la izquierda en modo bit
`>>=`	Asignación de desplazamiento a la derecha en modo bit
`>>>=`	Asignación de desplazamiento a la derecha en modo bit sin signo

Operador	Operación realizada
&=	Asignación de AND en modo bit
^=	Asignación de XOR en modo bit
|=	Asignación de OR en modo bit

Condicionales

ActionScript 3.0 proporciona tres sentencias condicionales básicas que se pueden usar para controlar el flujo del programa.

if..else

La sentencia condicional `if..else` permite comprobar una condición y ejecutar un bloque de código si dicha condición existe, o ejecutar un bloque de código alternativo si la condición no existe. Por ejemplo, el siguiente fragmento de código comprueba si el valor de `x` es superior a 20 y genera una función `trace()` en caso afirmativo o genera una función `trace()` diferente en caso negativo:

```
if (x > 20)
{
    trace("x is > 20");
}
else
{
    trace("x is <= 20");
}
```

Si no desea ejecutar un bloque de código alternativo, se puede utilizar la sentencia `if` sin la sentencia `else`.

if..else if

Puede comprobar varias condiciones utilizando la sentencia condicional `if..else if`. Por ejemplo, el siguiente fragmento de código no sólo comprueba si el valor de `x` es superior a 20, sino que también comprueba si el valor de `x` es negativo:

```
if (x > 20)
{
    trace("x is > 20");
}
else if (x < 0)
{
    trace("x is negative");
}
```

Si una sentencia `if` o `else` va seguida de una sola sentencia, no es necesario escribir dicha sentencia entre llaves. Por ejemplo, en el código siguiente no se usan llaves:

```
if (x > 0)
    trace("x is positive");
else if (x < 0)
    trace("x is negative");
else
    trace("x is 0");
```

No obstante, Adobe recomienda utilizar siempre llaves, ya que podría producirse un comportamiento inesperado si más adelante se añadieran sentencias a una sentencia condicional que no esté escrita entre llaves. Por ejemplo, en el código siguiente el valor de `positiveNums` aumenta en 1 independientemente de si la evaluación de la condición devuelve `true`:

```
var x:int;
var positiveNums:int = 0;

if (x > 0)
    trace("x is positive");
    positiveNums++;

trace(positiveNums); // 1
```

switch

La sentencia `switch` resulta útil si hay varios hilos de ejecución que dependen de la misma expresión de condición. La funcionalidad que proporciona es similar a una serie larga de sentencias `if..else if`, pero su lectura resulta un tanto más sencilla. En lugar de probar una condición para un valor booleano, la sentencia `switch` evalúa una expresión y utiliza el resultado para determinar el bloque de código que debe ejecutarse. Los bloques de código empiezan por una sentencia `case` y terminan con una sentencia `break`. Por ejemplo, la siguiente sentencia `switch` imprime el día de la semana en función del número de día devuelto por el método `Date.getDay()`:

```
var someDate:Date = new Date();
var dayNum:uint = someDate.getDay();
switch(dayNum)
{
    case 0:
        trace("Sunday");
        break;
    case 1:
        trace("Monday");
        break;
    case 2:
        trace("Tuesday");
        break;
    case 3:
        trace("Wednesday");
        break;
    case 4:
        trace("Thursday");
        break;
    case 5:
        trace("Friday");
        break;
    case 6:
        trace("Saturday");
        break;
    default:
        trace("Out of range");
        break;
}
```

Reproducir indefinidamente

Las sentencias de bucle permiten ejecutar un bloque específico de código repetidamente utilizando una serie de valores o variables. Adobe recomienda escribir siempre el bloque de código entre llaves ({}). Aunque puede omitir las llaves si el bloque de código sólo contiene una sentencia, no es recomendable que lo haga por la misma razón expuesta para las condicionales: aumenta la posibilidad de que las sentencias añadidas posteriormente se excluyan inadvertidamente del bloque de código. Si posteriormente se añade una sentencia que se desea incluir en el bloque de código, pero no se añaden las llaves necesarias, la sentencia no se ejecutará como parte del bucle.

for

El bucle `for` permite repetir una variable para un rango de valores específico. Debe proporcionar tres expresiones en una sentencia `for`: una variable que se establece con un valor inicial, una sentencia condicional que determina cuándo termina la reproducción en bucle y una expresión que cambia el valor de la variable con cada bucle. Por ejemplo, el siguiente código realiza cinco bucles. El valor de la variable `i` comienza en 0 y termina en 4, mientras que la salida son los números 0 a 4, cada uno de ellos en su propia línea.

```
var i:int;
for (i = 0; i < 5; i++)
{
    trace(i);
}
```

for..in

El bucle `for..in` recorre las propiedades de un objeto o los elementos de un conjunto. Por ejemplo, se puede utilizar un bucle `for..in` para recorrer las propiedades de un objeto genérico (las propiedades de un objeto no se guardan en ningún orden concreto, por lo que pueden aparecer en un orden aparentemente impredecible):

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj)
{
    trace(i + ": " + myObj[i]);
}
// output:
// x: 20
// y: 30
```

También se pueden recorrer los elementos de un conjunto:

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray)
{
    trace(myArray[i]);
}
// output:
// one
// two
// three
```

Lo que no se puede hacer es repetir las propiedades de un objeto si se trata de una instancia de una clase definida por el usuario, a no ser que la clase sea una clase dinámica. Incluso con instancias de clases dinámicas, sólo se pueden repetir las propiedades que se añadan dinámicamente.

for each..in

El bucle `for each..in` recorre los elementos de una colección, que puede estar formada por las etiquetas de un objeto XML o XMLList, los valores de las propiedades de un objeto o los elementos de un conjunto. Por ejemplo, como muestra el fragmento de código siguiente, el bucle `for each..in` se puede utilizar para recorrer las propiedades de un objeto genérico, pero al contrario de lo que ocurre con el bucle `for..in`, la variable de iteración de los bucles `for each..in` contiene el valor contenido por la propiedad en lugar del nombre de la misma:

```
var myObj:Object = {x:20, y:30};
for each (var num in myObj)
{
    trace(num);
}
// output:
// 20
// 30
```

Se puede recorrer un objeto XML o XMLList, como se indica en el siguiente ejemplo:

```
var myXML:XML = <users>
    <fname>Jane</fname>
    <fname>Susan</fname>
    <fname>John</fname>
</users>;

for each (var item in myXML.fname)
{
    trace(item);
}
/* output
Jane
Susan
John
*/
```

También se pueden recorrer los elementos de un conjunto, como se indica en este ejemplo:

```
var myArray:Array = ["one", "two", "three"];
for each (var item in myArray)
{
    trace(item);
}
// output:
// one
// two
// three
```

No se pueden recorrer las propiedades de un objeto si el objeto es una instancia de una clase cerrada. Tampoco se pueden recorrer las propiedades fijas (propiedades definidas como parte de una definición de clase), ni siquiera para las instancias de clases dinámicas.

while

El bucle `while` es como una sentencia `if` que se repite con tal de que la condición sea `true`. Por ejemplo, el código siguiente produce el mismo resultado que el ejemplo del bucle `for`:

```
var i:int = 0;
while (i < 5)
{
    trace(i);
    i++;
}
```

Una desventaja que presenta el uso de los bucles `while` frente a los bucles `for` es que es más probable escribir un bucle infinito con bucles `while`. El código de ejemplo de bucle `for` no se compila si se omite la expresión que aumenta la variable de contador, mientras que el ejemplo de bucle `while` sí se compila si se omite dicho paso. Sin la expresión que incrementa `i`, el bucle se convierte en un bucle infinito.

do..while

El bucle `do..while` es un bucle `while` que garantiza que el bloque de código se ejecuta al menos una vez, ya que la condición se comprueba después de que se ejecute el bloque de código. El código siguiente muestra un ejemplo simple de un bucle `do..while` que genera una salida aunque no se cumple la condición:

```
var i:int = 5;
do
{
    trace(i);
    i++;
} while (i < 5);
// output: 5
```

Funciones

Las *funciones* son bloques de código que realizan tareas específicas y pueden reutilizarse en el programa. Hay dos tipos de funciones en ActionScript 3.0: *métodos* y *cierres de función*. Llamar a una función método o cierre de función depende del contexto en el que se define la función. Una función se denomina método si se define como parte de una definición de clase o se asocia a una instancia de un objeto. Y se denomina cierre de función si se define de cualquier otra manera.

Las funciones siempre han sido muy importantes en ActionScript. Por ejemplo, en ActionScript 1.0 la palabra clave `class` no existía, por lo que las "clases" se definían mediante funciones constructoras. Aunque la palabra clave `class` se añadió posteriormente al lenguaje, sigue siendo importante comprender a fondo las funciones para aprovechar al máximo las capacidades del lenguaje. Esto puede ser difícil de entender para los programadores que esperan que las funciones de ActionScript se comporten de forma similar a las funciones de lenguajes como C++ o Java. Aunque una definición básica de función e invocación no debe resultar difícil para los programadores con experiencia, algunas de las características más avanzadas de las funciones de ActionScript requieren una explicación.

Fundamentos de la utilización de funciones

En esta sección se ofrece una definición básica de función y se describen las técnicas de invocación.

Invocación de funciones

Para llamar a una función se utiliza su identificador seguido del operador paréntesis (`()`). Se puede utilizar el operador paréntesis para escribir los parámetros de función que se desea enviar a la función. Por ejemplo, la función `trace()`, que es una función de nivel superior en ActionScript 3.0, se usa por todo el manual:

```
trace("Use trace to help debug your script");
```

Si se llama a una función sin parámetros, hay que utilizar un par de paréntesis vacíos. Por ejemplo, se puede utilizar el método `Math.random()`, que no admite parámetros, para generar un número aleatorio:

```
var randomNum:Number = Math.random();
```

Funciones definidas por el usuario

Hay dos formas de definir una función en ActionScript 3.0: se puede utilizar una sentencia de función o una expresión de función. La técnica que se elija dependerá de si se prefiere un estilo de programación más estático o más dinámico. Si se prefiere la programación estática, o en modo estricto, se deben definir las funciones con sentencias de función. Las funciones deben definirse con expresiones de función si existe la necesidad específica de hacerlo. Las expresiones de función se suelen usar en programación dinámica (en modo estándar).

Sentencias de función

Las sentencias de función son la técnica preferida para definir funciones en modo estricto. Una sentencia de función empieza con la palabra clave `function`, seguida de:

- El nombre de la función
- Los parámetros, en una lista delimitada por comas y escrita entre paréntesis
- El cuerpo de la función (es decir, el código ActionScript que debe ejecutarse cuando se invoca la función), escrito entre llaves

  Por ejemplo, el código siguiente crea una función que define un parámetro y después invoca la función con la cadena `"hello"` como valor del parámetro:

```
function traceParameter(aParam:String)
{
    trace(aParam);
}

traceParameter("hello"); // hello
```

Expresiones de función

La segunda manera de declarar una función es utilizar una sentencia de asignación con una expresión de función (también se suele llamar literal de función o función anónima). Éste es un método que requiere escribir más y que se usaba mucho en versiones anteriores de ActionScript.

Una sentencia de asignación con una expresión de función empieza por la palabra clave `var`, seguida de:

- El nombre de la función
- El operador dos puntos (`:`)
- La clase `Function` para indicar el tipo de datos
- El operador de asignación (`=`)
- La palabra clave `function`
- Los parámetros, en una lista delimitada por comas y escrita entre paréntesis
- El cuerpo de la función (es decir, el código ActionScript que debe ejecutarse cuando se invoca la función), escrito entre llaves

  Por ejemplo, el código siguiente declara la función `traceParameter` mediante una expresión de función:

```
var traceParameter:Function = function (aParam:String)
{
    trace(aParam);
};
traceParameter("hello"); // hello
```

Tenga en cuenta que, a diferencia de lo que ocurre en una sentencia de función, no se especifica un nombre de función. Otra diferencia importante entre las expresiones de función y las sentencias de función es que una expresión de función es una expresión, no una sentencia. Esto significa que una expresión de función no es independiente, como una sentencia de función. Una expresión de función sólo se puede utilizar como una parte de una sentencia (normalmente una sentencia de asignación). En el siguiente ejemplo se muestra la asignación de una expresión de función a un elemento de conjunto:

```
var traceArray:Array = new Array();
traceArray[0] = function (aParam:String)
{
    trace(aParam);
};
traceArray[0]("hello");
```

Criterios para elegir entre sentencias y expresiones

Como regla general, se debe utilizar una sentencia de función a menos que circunstancias específicas requieran una expresión. Las sentencias de función son menos detalladas y proporcionan una experiencia más uniforme entre el modo estricto y el modo estándar que las expresiones de función.

También son más fáciles de leer que las sentencias de asignación que contienen expresiones de función. Por otra parte, las sentencias de función hacen que el código sea más conciso; son menos confusas que las expresiones de función, que requieren utilizar las palabras clave `var` y `function`.

Además, proporcionan una experiencia más uniforme entre los dos modos de compilador, ya que permiten utilizar la sintaxis con punto en modo estándar y en modo estricto para invocar un método declarado con una sentencia de función. Esto no es así necesariamente para los métodos declarados con una expresión de función. Por ejemplo, el código siguiente define una clase denominada Example con dos métodos: `methodExpression()`, que se declara con una expresión de función, y `methodStatement()`, que se declara con una sentencia de función. En modo estricto no se puede utilizar la sintaxis con punto para invocar el método `methodExpression()`.

```
class Example
{
var methodExpression = function() {}
function methodStatement() {}
}

var myEx:Example = new Example();
myEx.methodExpression(); // error in strict mode; okay in standard mode
myEx.methodStatement(); // okay in strict and standard modes
```

Las expresiones de función se consideran más apropiadas para la programación centrada en el comportamiento dinámico (en tiempo de ejecución). Si se prefiere utilizar el modo estricto, pero también hay que llamar a un método declarado con una expresión de función, se puede utilizar cualquiera de las dos técnicas. En primer lugar, se puede llamar al método utilizando corchetes (`[]`) el lugar del operador punto (`.`). La siguiente llamada a método funciona correctamente tanto en modo estricto como en modo estándar:

```
myExample["methodLiteral"]();
```

En segundo lugar, se puede declarar toda la clase como una clase dinámica. Aunque esto permite llamar al método con el operador punto, la desventaja es que se sacrifica parte de la funcionalidad en modo estricto para todas las instancias de la clase. Por ejemplo, el compilador no genera un error si se intenta acceder a una propiedad no definida en una instancia de una clase dinámica.

Hay algunas circunstancias en las que las expresiones de función son útiles. Las expresiones de función se suelen utilizar para crear funciones que se utilizan una sola vez y después se descartan. Otro uso menos común es asociar una función a una propiedad de prototipo. Para obtener más información, consulte "El objeto prototype" en la página 123.

Hay dos diferencias sutiles entre las sentencias de función y las expresiones de función que se deben tener en cuenta al elegir la técnica que se va a utilizar. La primera diferencia es que las expresiones de función no existen de forma independiente como objetos con respecto a la administración de la memoria y la eliminación de datos innecesarios. Es decir, cuando se asigna una expresión de función a otro objeto, como un elemento de conjunto o una propiedad de objeto, se crea la única referencia a esa expresión de función en el código. Si el conjunto o el objeto al que la expresión de función está asociada se salen del ámbito o deja de estar disponible, se dejará de tener acceso a la expresión de función. Si se elimina el conjunto o el objeto, la memoria utilizada por la expresión de función quedará disponible para la eliminación de datos innecesarios, lo que significa que se podrá recuperar esa memoria y reutilizarla para otros propósitos.

En el siguiente ejemplo se muestra que, para una expresión de función, cuando se elimina la propiedad a la que está asignada la expresión, la función deja de estar disponible. La clase Test es dinámica, lo que significa que se puede añadir una propiedad denominada `functionExp` que contendrá una expresión de función. Se puede llamar a la función `functionExp()` con el operador punto, pero cuando se elimina la propiedad `functionExp`, la función deja de ser accesible.

```
dynamic class Test {}
var myTest:Test = new Test();

// function expression
myTest.functionExp = function () { trace("Function expression") };
myTest.functionExp(); // Function expression
delete myTest.functionExp;
myTest.functionExp(); // error
```

Si, por otra parte, la función se define primero con una sentencia de función, existe como su propio objeto y seguirá existiendo incluso después de que se elimine la propiedad a la que está asociada. El operador `delete` sólo funciona en propiedades de objetos, por lo que incluso una llamada para eliminar la función `stateFunc()` no funciona.

```
dynamic class Test {}
var myTest:Test = new Test();

// function statement
function stateFunc() { trace("Function statement") }
myTest.statement = stateFunc;
myTest.statement(); // Function statement
delete myTest.statement;
delete stateFunc; // no effect
stateFunc();// Function statement
myTest.statement(); // error
```

La segunda diferencia entre las sentencias de función y las expresiones de función es que las sentencias de función existen en todo el ámbito en que están definidas, incluso en sentencias que aparecen antes que la sentencia de función. En cambio, las expresiones de función sólo están definidas para las sentencias posteriores. Por ejemplo, el código siguiente llama correctamente a la función `scopeTest()` antes de que se defina:

```
statementTest(); // statementTest

function statementTest():void
{
    trace("statementTest");
}
```

Las expresiones de función no están disponibles antes de ser definidas, por lo que el código siguiente produce un error en tiempo de ejecución:

```
expressionTest(); // run-time error

var expressionTest:Function = function ()
{
    trace("expressionTest");
}
```

Devolución de valores de funciones

Para devolver un valor de la función se debe utilizar la sentencia `return` seguida de la expresión o el valor literal que se desea devolver. Por ejemplo, el código siguiente devuelve una expresión que representa al parámetro:

```
function doubleNum(baseNum:int):int
{
    return (baseNum * 2);
}
```

Tenga en cuenta que la sentencia `return` finaliza la función, por lo que las sentencias que estén por debajo de una sentencia `return` no se ejecutarán, como se indica a continuación:

```
function doubleNum(baseNum:int):int {
    return (baseNum * 2);
    trace("after return"); // This trace statement will not be executed.
}
```

En modo estricto se debe devolver un valor del tipo apropiado si se elige especificar un tipo devuelto. Por ejemplo, el código siguiente genera un error en modo estricto porque no devuelve un valor válido:

```
function doubleNum(baseNum:int):int
{
    trace("after return");
}
```

Funciones anidadas

Es posible anidar funciones, lo que significa que pueden declararse funciones dentro de otras funciones. Una función anidada sólo está disponible dentro de su función principal, a menos que se pase una referencia a la función a código externo. Por ejemplo, el código siguiente declara dos funciones anidadas dentro de la función `getNameAndVersion()`:

```
function getNameAndVersion():String
{
    function getVersion():String
    {
        return "10";
    }
    function getProductName():String
    {
        return "Flash Player";
    }
    return (getProductName() + " " + getVersion());
}
trace(getNameAndVersion()); // Flash Player 10
```

Cuando se pasan funciones anidadas a código externo, se pasan como cierres de función, lo que significa que la función retiene todas las definiciones que hubiera en el ámbito cuando se definió la función. Para obtener más información, consulte "Ámbito de una función" en la página 91.

Parámetros de función

ActionScript 3.0 proporciona funcionalidad para los parámetros de función que puede resultar novedosa para los programadores que empiecen a estudiar el lenguaje. Aunque la mayoría de los programadores deberían estar familiarizados con la idea de pasar parámetros por valor o referencia, es posible que el objeto `arguments` y el parámetro ... (rest) sean desconocidos para muchos.

Pasar argumentos por valor o por referencia

En muchos lenguajes de programación, es importante comprender la diferencia entre pasar argumentos por valor o por referencia; esta diferencia puede afectar a la manera de diseñar el código.

Al pasar por valor, el valor del argumento se copia en una variable local para usarlo en la función. Al pasar por referencia, sólo se pasa una referencia al argumento, en lugar del valor real. No se realiza ninguna copia del argumento real. En su lugar, se crea una referencia a la variable pasada como argumento y se asigna dicha referencia a una variable local para usarla en la función. Como una referencia a una variable externa a la función, la variable local proporciona la capacidad de cambiar el valor de la variable original.

En ActionScript 3.0, todos los argumentos se pasan por referencia, ya que todos los valores se almacenan como objetos. No obstante, los objetos que pertenecen a los tipos de datos simples, como Boolean, Number, int, uint y String, tienen operadores especiales que hacen que se comporten como si se pasaran por valor. Por ejemplo, el código siguiente crea una función denominada `passPrimitives()` que define dos parámetros denominados `xParam` y `yParam`, ambos de tipo int. Estos parámetros son similares a variables locales declaradas en el cuerpo de la función `passPrimitives()`. Cuando se llama a la función con los argumentos `xValue` e `yValue`, los parámetros `xParam` e `yParam` se inicializan con referencias a los objetos int representados por `xValue` e `yValue`. Como los argumentos son valores simples, se comportan como si se pasaran por valor. Aunque `xParam` e `yParam` sólo contienen inicialmente referencias a los objetos `xValue` e `yValue`, los cambios realizados a las variables en el cuerpo de la función generan nuevas copias de los valores en la memoria.

```
function passPrimitives(xParam:int, yParam:int):void
{
    xParam++;
    yParam++;
    trace(xParam, yParam);
}

var xValue:int = 10;
var yValue:int = 15;
trace(xValue, yValue);// 10 15
passPrimitives(xValue, yValue); // 11 16
trace(xValue, yValue);// 10 15
```

En la función `passPrimitives()`, los valores de `xParam` e `yParam` se incrementan, pero esto no afecta a los valores de `xValue` e `yValue`, como se indica en la última sentencia `trace`. Esto es así aunque se asigne a los parámetros los mismos nombres que a las variables, `xValue` e `yValue`, ya que dentro de la función `xValue` e `yValue` señalarían nuevas ubicaciones de la memoria que existen por separado de las variables externas a la función que tienen el mismo nombre.

Todos los demás objetos (es decir, los objetos que no pertenecen a los tipos de datos simples) se pasan siempre por referencia, ya que esto ofrece la capacidad de cambiar el valor de la variable original. Por ejemplo, el código siguiente crea un objeto denominado `objVar` con dos propiedades, `x` e `y`. El objeto se pasa como un argumento a la función `passByRef()`. Como el objeto no es un tipo simple, no sólo se pasa por referencia, sino que también se mantiene como una referencia. Esto significa que los cambios realizados en los parámetros dentro de la función afectarán a las propiedades del objeto fuera de la función.

```
function passByRef(objParam:Object):void
{
    objParam.x++;
    objParam.y++;
    trace(objParam.x, objParam.y);
}
var objVar:Object = {x:10, y:15};
trace(objVar.x, objVar.y); // 10 15
passByRef(objVar); // 11 16
trace(objVar.x, objVar.y); // 11 16
```

El parámetro `objParam` hace referencia al mismo objeto que la variable `objVar` global. Como se puede ver en las sentencias `trace` del ejemplo, los cambios realizados en las propiedades `x` e `y` del objeto `objParam` se reflejan en el objeto `objVar`.

Valores predeterminados de los parámetros

En ActionScript 3.0 se incluye como novedad la capacidad de declarar *valores predeterminados de parámetros* para una función. Si una llamada a una función con valores predeterminados de parámetros omite un parámetro con valores predeterminados, se utiliza el valor especificado en la definición de la función para ese parámetro. Todos los parámetros con valores predeterminados deben colocarse al final de la lista de parámetros. Los valores asignados como valores predeterminados deben ser constantes de tiempo de compilación. La existencia de un valor predeterminado para un parámetro convierte de forma efectiva a ese parámetro en un *parámetro opcional*. Un parámetro sin un valor predeterminado se considera un *parámetro requerido*.

Por ejemplo, el código siguiente crea una función con tres parámetros, dos de los cuales tienen valores predeterminados. Cuando se llama a la función con un solo parámetro, se utilizan los valores predeterminados de los parámetros.

```
function defaultValues(x:int, y:int = 3, z:int = 5):void
{
    trace(x, y, z);
}
defaultValues(1); // 1 3 5
```

El objeto arguments

Cuando se pasan parámetros a una función, se puede utilizar el objeto `arguments` para acceder a información sobre los parámetros pasados a la función. Algunos aspectos importantes del objeto `arguments` son:

- El objeto `arguments` es un conjunto que incluye todos los parámetros pasados a la función.
- La propiedad `arguments.length` notifica el número de parámetros pasados a la función.
- La propiedad `arguments.callee` proporciona una referencia a la misma función, que resulta útil para llamadas recursivas a expresiones de función.

  Nota: *el objeto `arguments` no estará disponible si algún parámetro tiene el nombre `arguments` o si se utiliza el parámetro ... (rest).*

  Si se hace referencia al objeto `arguments` en el cuerpo de una función, ActionScript 3.0 permite que las llamadas a funciones incluyan más parámetros que los definidos en la definición de la función, pero generará un error del compilador en modo estricto si el número de parámetros no coincide con el número de parámetros requeridos (y de forma opcional, los parámetros opcionales). Se puede utilizar el conjunto aspect del objeto `arguments` para acceder a cualquier parámetro pasado a la función, independientemente de si ese parámetro está definido en la definición de la función. En el ejemplo siguiente, que sólo compila en modo estándar, se utiliza el conjunto `arguments` junto con la propiedad `arguments.length` para hacer un seguimiento de todos los parámetros pasados a la función `traceArgArray()`:

  ```
  function traceArgArray(x:int):void
  {
      for (var i:uint = 0; i < arguments.length; i++)
      {
          trace(arguments[i]);
      }
  }

  traceArgArray(1, 2, 3);

  // output:
  // 1
  // 2
  // 3
  ```

  La propiedad `arguments.callee` se suele utilizar en funciones anónimas para crear recursión. Se puede utilizar para añadir flexibilidad al código. Si el nombre de una función recursiva cambia a lo largo del ciclo de desarrollo, no es necesario preocuparse de cambiar la llamada recursiva en el cuerpo de la función si se utiliza `arguments.callee` en lugar del nombre de la función. La propiedad `arguments.callee` se utiliza en la siguiente expresión de función para habilitar la recursión:

```
var factorial:Function = function (x:uint)
{
    if(x == 0)
    {
        return 1;
    }
    else
    {
        return (x * arguments.callee(x - 1));
    }
}

trace(factorial(5)); // 120
```

Si se utiliza el parámetro ... (rest) en la declaración de la función, el objeto `arguments` no estará disponible. Hay que acceder a los parámetros a través de los nombres de parámetro declarados.

También hay que procurar no utilizar la cadena `"arguments"` como nombre de parámetro, ya que ocultará el objeto `arguments`. Por ejemplo, si se vuelve a escribir la función `traceArgArray()` de forma que se añade un parámetro `arguments`, las referencias a `arguments` en el cuerpo de la función hacen referencia al parámetro, en lugar de al objeto `arguments`. El siguiente código no produce ningún resultado:

```
function traceArgArray(x:int, arguments:int):void
{
    for (var i:uint = 0; i < arguments.length; i++)
    {
        trace(arguments[i]);
    }
}

traceArgArray(1, 2, 3);

// no output
```

El objeto `arguments` de versiones anteriores de ActionScript también contenía una propiedad denominada `caller`, que es una referencia a la función que llamó a la función actual. La propiedad `caller` no existe en ActionScript 3.0, pero si se necesita una referencia a la función que llama, se puede modificar dicha función de forma que pase un parámetro adicional que sea una referencia sí mismo.

El parámetro ...(rest)

ActionScript 3.0 introduce una declaración de un parámetro nuevo que se llama ... (rest). Este parámetro permite especificar un parámetro de tipo conjunto que acepta un número arbitrario de argumentos delimitados por comas. El parámetro puede tener cualquier nombre que no sea una palabra reservada. Este parámetro debe especificarse el último. El uso de este parámetro hace que el objeto `arguments` no esté disponible. Aunque el parámetro ... (rest) ofrece la misma funcionalidad que el conjunto `arguments` y la propiedad `arguments.length`, no proporciona funcionalidad similar a la que ofrece `arguments.callee`. Hay que asegurarse de que no es necesario utilizar `arguments.callee` antes de utilizar el parámetro ... (rest).

En el ejemplo siguiente se reescribe la función `traceArgArray()` con el parámetro ... (rest) en lugar del objeto `arguments`:

```
function traceArgArray(... args):void
{
    for (var i:uint = 0; i < args.length; i++)
    {
        trace(args[i]);
    }
}

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

El parámetro ... (rest) también puede utilizarse con otros parámetros, con tal de que sea el último parámetro de la lista. En el ejemplo siguiente se modifica la función `traceArgArray()` de forma que su primer parámetro, `x`, sea de tipo int y el segundo parámetro utilice el parámetro ... (rest). La salida omite el primer valor porque el primer parámetro ya no forma parte del conjunto creada por el parámetro ... (rest).

```
function traceArgArray(x: int, ... args)
{
    for (var i:uint = 0; i < args.length; i++)
    {
        trace(args[i]);
    }
}

traceArgArray(1, 2, 3);

// output:
// 2
// 3
```

Funciones como objetos

En ActionScript 3.0 las funciones son objetos. Al crear una función, se crea un objeto que no sólo se puede pasar como un parámetro a otra función, sino que además tiene propiedades y métodos asociados.

Las funciones pasadas como argumentos a otra función se pasan por referencia, no por valor. Al pasar una función como un argumento sólo se utiliza el identificador y no el operador paréntesis que se utiliza para llamar al método. Por ejemplo, el código siguiente pasa una función denominada `clickListener()` como un argumento al método `addEventListener()`:

```
addEventListener(MouseEvent.CLICK, clickListener);
```

El método `Array.sort()` también define un parámetro que acepta una función. Para ver un ejemplo de una función de ordenación personalizada que se utiliza como argumento de la función `Array.sort()`, consulte “Ordenación de un conjunto” en la página 167.

Aunque pueda parecer extraño a los programadores sin experiencia en ActionScript, las funciones pueden tener propiedades y métodos, igual que cualquier otro objeto. De hecho, cada función tiene una propiedad de sólo lectura denominada `length` que almacena el número de parámetros definidos para la función. Es distinta de la propiedad `arguments.length`, que notifica el número de argumentos enviados a la función. Debe recordarse que en ActionScript el número de argumentos enviados a una función pueden superar el número de parámetros definidos para dicha función. En el ejemplo siguiente, que sólo se compila en modo estándar porque el modo estricto requiere una coincidencia exacta entre el número de argumentos pasados y el número de parámetros definidos, se muestra la diferencia entre las dos propiedades:

```
// Compiles only in standard mode
function traceLength(x:uint, y:uint):void
{
    trace("arguments received: " + arguments.length);
    trace("arguments expected: " + traceLength.length);
}

traceLength(3, 5, 7, 11);
/* output:
arguments received: 4
arguments expected: 2 */
```

En modo estándar se pueden definir propiedades de función propias fuera del cuerpo de la función. Las propiedades de función pueden servir como propiedades casi estáticas que permiten guardar el estado de una variable relacionada con la función. Por ejemplo, si se desea hacer un seguimiento del número de veces que se llama a una función determinada. Esta funcionalidad puede ser útil cuando se programa un juego y se desea hacer un seguimiento del número de veces que un usuario utiliza un comando específico, aunque también se podría utilizar una propiedad de clase estática para esto. El ejemplo siguiente, que sólo compila en modo estándar porque el modo estricto no permite añadir propiedades dinámicas a funciones, crea una propiedad de función fuera de la declaración de función e incrementa la propiedad cada vez que se llama a la función:

```
// Compiles only in standard mode
var someFunction:Function = function ():void
{
    someFunction.counter++;
}

someFunction.counter = 0;

someFunction();
someFunction();
trace(someFunction.counter); // 2
```

Ámbito de una función

El ámbito de una función determina no sólo en qué partes de un programa se puede llamar a esa función, sino también a qué definiciones tiene acceso la función. Las mismas reglas de ámbito que se aplican a los identificadores de variable se aplican a los identificadores de función. Una función declarada en el ámbito global estará disponible en todo el código. Por ejemplo, ActionScript 3.0 contiene funciones globales, como `isNaN()` y `parseInt()`, que están disponibles desde cualquier punto del código. Una función anidada (una función declarada dentro de otra función) puede utilizarse en cualquier punto de la función en que se declaró.

La cadena de ámbitos

Cuando se inicia la ejecución de una función, se crean diversos objetos y propiedades. En primer lugar, se crea un objeto especial denominado *objeto de activación* que almacena los parámetros y las variables o funciones locales declaradas en el cuerpo de la función. No se puede acceder al objeto de activación directamente, ya que es un mecanismo interno. En segundo lugar, se crea una *cadena de ámbitos* que contiene una lista ordenada de objetos en las que Flash Player o Adobe AIR comprueba las declaraciones de identificadores. Cada función que ejecuta tiene una cadena de ámbitos que se almacena en una propiedad interna. Para una función anidada, la cadena de ámbitos empieza en su propio objeto de activación, seguido del objeto de activación de la función principal. La cadena continúa de esta manera hasta que llega al objeto global. El objeto global se crea cuando se inicia un programa de ActionScript y contiene todas las variables y funciones globales.

Cierres de función

Un *cierre de función* es un objeto que contiene una instantánea de una función y su *entorno léxico*. El entorno léxico de una función incluye todas las variables, propiedades, métodos y objetos de la cadena de ámbitos de la función, junto con sus valores. Los cierres de función se crean cada vez que una función se ejecuta aparte de un objeto o una clase. El hecho de que los cierres de función conserven el ámbito en que se definieron crea resultados interesantes cuando se pasa una función como un argumento o un valor devuelto en un ámbito diferente.

Por ejemplo, el código siguiente crea dos funciones: `foo()`, que devuelve una función anidada denominada `rectArea()` que calcula el área de un rectángulo y `bar()`, que llama a `foo()` y almacena el cierre de función devuelto en una variable denominada `myProduct`. Aunque la función `bar()` define su propia variable local `x` (con el valor 2), cuando se llama al cierre de función `myProduct()`, conserva la variable `x` (con el valor 40) definida en la función `foo()`. Por tanto, la función `bar()` devuelve el valor `160` en lugar de`8`.

```
function foo():Function
{
    var x:int = 40;
    function rectArea(y:int):int // function closure defined
    {
        return x * y
    }
    return rectArea;
}
function bar():void
{
    var x:int = 2;
    var y:int = 4;
    var myProduct:Function = foo();
    trace(myProduct(4)); // function closure called
}
bar(); // 160
```

Los métodos se comportan de manera similar, ya que también conservan información sobre el entorno léxico en que se crearon. Esta característica se aprecia especialmente cuando se extrae un método de su instancia, lo que crea un método vinculado. La diferencia principal entre un cierre de función y un método vinculado es que el valor de la palabra clave `this` en un método vinculado siempre hace referencia a la instancia a la que estaba asociado originalmente, mientras que en un cierre de función el valor de la palabra clave `this` puede cambiar. Para obtener más información, consulte “Métodos” en la página 101.

Capítulo 5: Programación orientada a objetos con ActionScript

En este capítulo se describen los elementos de ActionScript que admiten la programación orientada a objetos. No se describen principios generales de la programación orientada a objetos, como el diseño de objetos, la abstracción, la encapsulación, la herencia y el polimorfismo. El capítulo se centra en la manera de aplicar estos principios con ActionScript 3.0.

Como ActionScript fue originalmente un lenguaje de creación de scripts, en ActionScript 3.0 la compatibilidad con la programación orientada a objetos es opcional. Esto proporciona a los programadores la flexibilidad de elegir el mejor enfoque para proyectos de ámbito y complejidad variables. Para tareas pequeñas, es posible que sea suficiente con utilizar un paradigma de programación mediante procedimientos. Para proyectos más grandes, los principios de la programación orientada a objetos pueden facilitar la comprensión, el mantenimiento y la ampliación del código.

Fundamentos de la programación orientada a objetos

Introducción a la programación orientada a objetos

La programación orientada a objetos es una forma de organizar el código de un programa agrupándolo en objetos, que son elementos individuales que contienen información (valores de datos) y funcionalidad. La utilización de un enfoque orientado a objetos para organizar un programa permite agrupar partes específicas de la información (por ejemplo, información de una canción como el título de álbum, el título de la pista o el nombre del artista) junto con funcionalidad o acciones comunes asociadas con dicha información (como "añadir pista a la lista de reproducción" o "reproducir todas las canciones de este artista"). Estos elementos se combinan en un solo elemento, denominado objeto (por ejemplo, un objeto "Album" o "MusicTrack"). Poder agrupar estos valores y funciones proporciona varias ventajas, como la capacidad de hacer un seguimiento de una sola variable en lugar de tener que controlar varias variables, agrupar funcionalidad relacionada y poder estructurar programas de maneras que reflejen mejor el mundo real.

Tareas comunes de la programación orientada a objetos

En la práctica, la programación orientada a objetos consta de dos partes. Por un lado, las estrategias y técnicas para diseñar un programa (*diseño orientado a objetos*). Esto es un tema amplio que no se tratará en este capítulo. El otro aspecto de la programación orientada a objetos son las estructuras de programación que están disponibles en un lenguaje de programación determinado para crear un programa con un enfoque orientado a objetos. En este capítulo se describen las siguientes tareas comunes de la programación orientada a objetos:

- Definición de clases
- Crear propiedades, métodos y descriptores de acceso get y set (métodos descriptores de acceso)
- Controlar el acceso a clases, propiedades, métodos y descriptores de acceso
- Crear propiedades y métodos estáticos
- Crear estructuras de tipo enumeración
- Definir y utilizar interfaces
- Trabajo con herencia, incluida la sustitución de elementos de clase

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Atributo: característica asignada a un elemento de clase (como una propiedad o un método) en la definición de clase. Los atributos se suelen utilizar para definir si la propiedad o el método serán accesibles para el código de otras partes del programa. Por ejemplo, `private` y `public` son atributos. Sólo se puede llamar a un método privado desde dentro de la clase; en cambio, se puede llamar a un método público desde cualquier parte del programa.
- Clase: definición de la estructura y el comportamiento de objetos de un tipo determinado (como una plantilla o un plano de los objetos de ese tipo de datos).
- Jerarquía de clases: estructura de varias clases relacionadas, que especifica qué clases heredan funcionalidad de otras clases.
- Constructor: método especial que se puede definir en una clase y que se llama para crear una instancia de la clase. Se suele utilizar un constructor para especificar valores predeterminados o realizar operaciones de configuración del objeto.
- Tipo de datos: tipo de información que una variable concreta puede almacenar. En general, *tipo de datos* significa lo mismo que *clase*.
- Operador punto: signo del punto (.), que en ActionScript y en muchos otros lenguajes de programación se utiliza para indicar que un nombre hace referencia a un elemento secundario de un objeto (como una propiedad o un método). Por ejemplo, en la expresión `myObject.myProperty`, el operador punto indica que el término `myProperty` hace referencia a algún valor que es un elemento del objeto denominado `myObject`.
- Enumeración: conjunto de valores de constante relacionados, agrupados por comodidad como propiedades de una clase individual.
- Herencia: mecanismo de la programación orientada a objetos que permite a una definición de clase incluir toda la funcionalidad de una definición de clase distinta (y añadir nueva funcionalidad).
- Instancia: objeto real creado en un programa.
- Espacio de nombres: fundamentalmente se trata de un atributo personalizado que ofrece un control más preciso sobre el código al que puede acceder otro código.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Como los listados de código de este capítulo se centran principalmente en definir y manipular tipos de datos, para probar los ejemplos hay que crear una instancia de la clase que se está definiendo, manipular esa instancia con sus propiedades o métodos y, a continuación, ver los valores de las propiedades de la instancia. Para ver esos valores hay que escribir valores en una instancia de campo de texto del escenario o utilizar la función `trace()` para imprimir valores en el panel Salida. Estas técnicas se describen detalladamente en "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

Clases

Una clase es una representación abstracta de un objeto. Una clase almacena información sobre los tipos de datos que un objeto puede contener y los comportamientos que un objeto puede exhibir. La utilidad de esta abstracción puede no ser apreciable al escribir scripts sencillos que sólo contienen unos pocos objetos que interactúan entre sí. Sin embargo, a medida que el ámbito de un programa crece y aumenta el número de objetos que hay que administrar, las clases pueden ayudar a obtener mayor control sobre la creación y la interacción mutua de los objetos.

Con ActionScript 1.0 los programadores podían utilizar objetos Function para crear construcciones similares a las clases. ActionScript 2.0 añadió formalmente la compatibilidad con las clases con palabras clave como `class` y `extends`. En ActionScript 3.0 no sólo se mantienen las palabras clave introducidas en ActionScript 2.0, sino que también se han añadido algunas capacidades nuevas, como el control de acceso mejorado con los atributos `protected` e `internal`, y mejor control de la herencia con las palabras clave `final` y `override`.

A los programadores con experiencia en la creación de clases con lenguajes de programación como Java, C++ o C#, el sistema de objetos de ActionScript les resultará familiar. ActionScript comparte muchas de las mismas palabras clave y nombres de atributo, como `class`, `extends` y `public`, que se describen en las siguiente secciones.

Nota: *en este capítulo, el término propiedad designa cualquier miembro de un objeto o una clase, incluidas variables, constantes y métodos. Por otra parte, aunque los términos class y static se suelen utilizar como sinónimos, en este capítulo tienen un significado diferente. Por ejemplo, en este capítulo la frase propiedades de clase hace referencia a todos los miembros de una clase, no únicamente a los miembros estáticos.*

Definiciones de clase

En las definiciones de clase de ActionScript 3.0 se utiliza una sintaxis similar a la utilizada en las definiciones de clase de ActionScript 2.0. La sintaxis correcta de una definición de clase requiere la palabra clave `class` seguida del nombre de la clase. El cuerpo de la clase, que se escribe entre llaves ({}), sigue al nombre de la clase. Por ejemplo, el código siguiente crea una clase denominada Shape que contiene una variable denominada `visible`:

```
public class Shape
{
    var visible:Boolean = true;
}
```

Un cambio de sintaxis importante afecta a las definiciones de clase que están dentro de un paquete. En ActionScript 2.0, si una clase estaba dentro de un paquete, había que incluir el nombre de paquete en la declaración de la clase. En ActionScript 3.0 se incluye la sentencia `package` y hay que incluir el nombre del paquete en la declaración del paquete, no en la declaración de clase. Por ejemplo, las siguientes declaraciones de clase muestran la manera de definir la clase BitmapData, que forma parte del paquete flash.display, en ActionScript 2.0 y en ActionScript 3.0:

```
// ActionScript 2.0
class flash.display.BitmapData {}

// ActionScript 3.0
package flash.display
{
    public class BitmapData {}
}
```

Atributos de clase

ActionScript 3.0 permite modificar las definiciones de clase mediante uno de los cuatro atributos siguientes:

Atributo	Definición
`dynamic`	Permite añadir propiedades a instancias en tiempo de ejecución.
`final`	No debe ser ampliada por otra clase.
`internal` (valor predeterminado)	Visible para referencias dentro del paquete actual.
`public`	Visible para referencias en todas partes.

Todos estos atributos, salvo `internal`, deben ser incluidos explícitamente para obtener el comportamiento asociado. Por ejemplo, si no se incluye el atributo `dynamic` al definir una clase, no se podrá añadir propiedades a una instancia de clase en tiempo de ejecución. Un atributo se asigna explícitamente colocándolo al principio de la definición de clase, como se muestra en el código siguiente:

```
dynamic class Shape {}
```

Hay que tener en cuenta que la lista no incluye un atributo denominado `abstract`. Esto se debe a que las clases abstractas no se admiten en ActionScript 3.0. También se debe tener en cuenta que la lista no incluye atributos denominados `private` y `protected`. Estos atributos sólo tienen significado dentro de una definición de clase y no se pueden aplicar a las mismas clases. Si no se desea que una clase sea visible públicamente fuera de un paquete, debe colocarse la clase dentro de un paquete y marcarse con el atributo `internal`. Como alternativa, se pueden omitir los atributos `internal` y `public`, y el compilador añadirá automáticamente el atributo `internal`. Si no se desea que una clase sea visible fuera del archivo de código fuente en el que está definida, se debe colocar al final del archivo de código fuente después de la llave final de la definición de paquete.

Cuerpo de la clase

El cuerpo de la clase, que se escribe entre llaves, se usa para definir las variables, constantes y métodos de la clase. En el siguiente ejemplo se muestra la declaración para la clase Accessibility en la API de Adobe Flash Player:

```
public final class Accessibility
{
    public static function get active():Boolean;
    public static function updateProperties():void;
}
```

También se puede definir un espacio de nombres dentro de un cuerpo de clase. En el siguiente ejemplo se muestra cómo se puede definir un espacio de nombres en el cuerpo de una clase y utilizarse como atributo de un método en dicha clase:

```
public class SampleClass
{
    public namespace sampleNamespace;
    sampleNamespace function doSomething():void;
}
```

ActionScript 3.0 permite incluir en el cuerpo de una clase no sólo definiciones, sino también sentencias. Las sentencias que están dentro de un cuerpo de clase pero fuera de una definición de método se ejecutan una sola vez: cuando se encuentra por primera vez la definición de la clase y se crea el objeto de clase asociado. En el ejemplo siguiente se incluye una llamada a una función externa, `hello()`, y una sentencia `trace` que emite un mensaje de confirmación cuando se define la clase:

```
function hello():String
{
    trace("hola");
}
class SampleClass
{
    hello();
    trace("class created");
}
// output when class is created
hola
class created
```

A diferencia de las versiones anteriores de ActionScript, en ActionScript 3.0 se permite definir en un mismo cuerpo de clase una propiedad estática y una propiedad de instancia con el mismo nombre. Por ejemplo, el código siguiente declara una variable estática denominada `message` y una variable de instancia con el mismo nombre:

```
class StaticTest
{
    static var message:String = "static variable";
    var message:String = "instance variable";
}
// In your script
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // output: static variable
trace(myST.message); // output: instance variable
```

Atributos de propiedad de clase

En las descripciones del modelo de objetos de ActionScript, el término *propiedad* significa cualquier cosa que pueda ser un miembro de una clase, incluidas variables, constantes y métodos. Esto difiere de la manera en que se utiliza el término en la Referencia del lenguaje y componentes ActionScript 3.0, donde se aplica a un concepto menos amplio y sólo incluye miembros de clase que son variables o se definen mediante un método captador o definidor. En ActionScript 3.0 hay un conjunto de atributos que se pueden utilizar con cualquier propiedad de una clase. En la tabla siguiente se muestra este conjunto de atributos.

Atributo	Definición
`internal` (valor predeterminado)	Visible para referencias dentro del mismo paquete.
`private`	Visible para referencias dentro de la misma clase.
`protected`	Visible para referencias en la misma clase y en clases derivadas.
`public`	Visible para referencias en todas partes.
`static`	Especifica que una propiedad pertenece a la clase en lugar de a las instancias de la clase.
`UserDefinedNamespace`	Nombre de espacio de nombres personalizado definido por el usuario.

Atributos del espacio de nombres de control de acceso

ActionScript 3.0 proporciona cuatro atributos especiales que controlan el acceso a las propiedades definidas dentro de una clase: `public`, `private`, `protected` e `internal`.

El atributo `public` hace que una propiedad esté visible en cualquier parte del script. Por ejemplo, para hacer que un método esté disponible para el código fuera de su paquete, hay que declarar el método con el atributo `public`. Esto se cumple para cualquier propiedad, independientemente de que se declare con la palabra clave `var`, `const` o `function`.

El atributo `private` hace que una propiedad sólo esté visible para los orígenes de llamada de la clase en la que se define la propiedad. Este comportamiento difiere del comportamiento del atributo `private` en ActionScript 2.0, que permitía a una subclase tener acceso a una propiedad privada de una superclase. Otro cambio importante de comportamiento está relacionado con el acceso en tiempo de ejecución. En ActionScript 2.0, la palabra clave `private` sólo prohibía el acceso en tiempo de compilación y se podía evitar fácilmente en tiempo de ejecución. Esto ya no se cumple en ActionScript 3.0. Las propiedades marcadas como `private` no están disponibles en tiempo de compilación ni en tiempo de ejecución.

Por ejemplo, el código siguiente crea una clase simple denominada PrivateExample con una variable privada y después intenta acceder a la variable privada desde fuera de la clase. En ActionScript 2.0, el acceso en tiempo de compilación estaba prohibido, pero la prohibición se podía evitar fácilmente utilizando el operador de acceso a una propiedad (`[]`), que realiza la búsqueda de propiedades en tiempo de ejecución, no en tiempo de compilación.

```
class PrivateExample
{
    private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample["privVar"]); // ActionScript 2.0 allows access, but in ActionScript 3.0, this
is a run-time error.
```

En ActionScript 3.0, un intento de acceder a una propiedad privada mediante el operador punto (`myExample.privVar`) provoca un error de tiempo de compilación si se utiliza el modo estricto. De lo contrario, el error se notifica en tiempo de ejecución, de la misma que manera que al usar el operador de acceso a una propiedad (`myExample["privVar"]`).

En la tabla siguiente se resumen los resultados de intentar acceder a una propiedad privada que pertenece a una clase cerrada (no dinámica):

	Modo estricto	**Modo estándar**
operador punto (`.`)	error en tiempo de compilación	error en tiempo de ejecución
operador corchete (`[]`)	error en tiempo de ejecución	error en tiempo de ejecución

En clases declaradas con el atributo `dynamic`, los intentos de acceder a una variable privada no provocarán un error en tiempo de ejecución. La variable simplemente no está visible, por lo que Flash Player o Adobe® AIR™ devuelven el valor `undefined`. No obstante, se producirá un error en tiempo de compilación si se utiliza el operador punto en modo estricto. El ejemplo siguiente es igual que el anterior, con la diferencia de que la clase PrivateExample se declara como una clase dinámica:

```
dynamic class PrivateExample
{
    private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample["privVar"]); // output: undefined
```

Las clases dinámicas generalmente devuelven el valor `undefined` en lugar de generar un error cuando código externo a una clase intenta acceder a una propiedad privada. En la tabla siguiente se muestra que sólo se genera un error cuando se utiliza el operador punto para acceder a una propiedad privada en modo estricto:

	Modo estricto	**Modo estándar**
operador punto (`.`)	error en tiempo de compilación	`undefined`
operador corchete (`[]`)	`undefined`	`undefined`

El atributo `protected`, que es una de las novedades de ActionScript 3.0, hace que una propiedad esté visible para los orígenes de llamada en su propia clase o en una subclase. Es decir, una propiedad protegida está disponible en su propia clase o para clases de nivel inferior en la jerarquía de herencia. Esto se cumple tanto si la subclase está en el mismo paquete como si está en un paquete diferente.

Para los usuarios familiarizados con ActionScript 2.0, esta funcionalidad es similar al atributo `private` en ActionScript 2.0. El atributo `protected` de ActionScript 3.0 también es similar al atributo `protected` en Java, pero difiere en que la versión de Java también permite acceder a quien realiza la llamada en el mismo paquete. El atributo `protected` resulta útil cuando se tiene una variable o método requerido por las subclases que se desea ocultar del código que esté fuera de la cadena de herencia.

El atributo `internal`, que es una de las novedades de ActionScript 3.0, hace que una propiedad esté visible para los orígenes de llamada en su propio paquete. Es el atributo predeterminado para el código de un paquete y se aplica a cualquier propiedad que no tenga ninguno de los siguientes atributos:

- `public`
- `private`
- `protected`
- un espacio de nombres definido por el usuario

El atributo `internal` es similar al control de acceso predeterminado en Java, aunque en Java no hay ningún nombre explícito para este nivel de acceso y sólo se puede alcanzar mediante la omisión de cualquier otro modificador de acceso. El atributo `internal` está disponible en ActionScript 3.0 para ofrecer la opción de indicar explícitamente la intención de hacer que una propiedad sólo sea visible para orígenes de llamada de su propio paquete.

Atributo static

El atributo `static`, que se puede utilizar con propiedades declaradas con las palabras clave `var`, `const` o `function`, permite asociar una propiedad a la clase en lugar de asociarla a instancias de la clase. El código externo a la clase debe llamar a propiedades estáticas utilizando el nombre de la clase en lugar de un nombre de instancia.

Las subclases no heredan las propiedades estáticas, pero las propiedades forman parte de una cadena de ámbitos de subclase. Esto significa que en el cuerpo de una subclase se puede utilizar una variable o un método estático sin hacer referencia a la clase en la que se definió. Para obtener más información, consulte "Propiedades estáticas no heredadas" en la página 117.

Atributos de espacio de nombres definido por el usuario

Como alternativa a los atributos de control de acceso predefinidos se puede crear un espacio de nombres personalizado para usarlo como un atributo. Sólo se puede utilizar un atributo de espacio de nombres por cada definición y no se puede utilizar en combinación con uno de los atributos de control de acceso (`public`, `private`, `protected`, `internal`). Para más información sobre el uso de espacios de nombres, consulte"Espacios de nombres" en la página 44.

Variables

Las variables pueden declararse con las palabras clave `var` o `const`. Es posible cambiar los valores de las variables declaradas con la palabra clave `var` varias veces durante la ejecución de un script. Las variables declaradas con la palabra clave `const` se denominan *constantes* y se les puede asignar valores una sola vez. Un intento de asignar un valor nuevo a una constante inicializada provoca un error. Para obtener más información, consulte "Constantes" en la página 70.

Variables estáticas

Las variables estáticas se declaran mediante una combinación de la palabra clave `static` y la sentencia `var` o `const`. Las variables estáticas, que se asocian a una clase en lugar de a una instancia de una clase, son útiles para almacenar y compartir información que se aplica a toda una clase de objetos. Por ejemplo, una variable estática es adecuada si se desea almacenar un recuento del número de veces que se crea una instancia de una clase o si se desea almacenar el número máximo de instancias de la clase permitidas.

En el ejemplo siguiente se crea una variable `totalCount` para hacer un seguimiento del número de instancias de clase creadas y una constante `MAX_NUM` para almacenar el número máximo de instancias creadas. Las variables `totalCount` y `MAX_NUM` son estáticas porque contienen valores que se aplican a la clase como un todo en lugar de a una instancia concreta.

```
class StaticVars
{
    public static var totalCount:int = 0;
    public static const MAX_NUM:uint = 16;
}
```

El código externo a la clase StaticVars y a cualquiera de sus subclases sólo puede hacer referencia a las propiedades `totalCount` y `MAX_NUM` a través de la misma clase. Por ejemplo, el siguiente código funciona:

```
trace(StaticVars.totalCount); // output: 0
trace(StaticVars.MAX_NUM); // output: 16
```

No se puede acceder a variables estáticas a través de una instancia de la clase, por lo que el código siguiente devuelve errores:

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // error
trace(myStaticVars.MAX_NUM); // error
```

Las variables declaradas con las palabras clave `static` y `const` deben ser inicializadas a la vez que se declara la constante, como hace la clase StaticVars para `MAX_NUM`. No se puede asignar un valor a `MAX_NUM` dentro del constructor o de un método de instancia. El código siguiente generará un error, ya que no es una forma válida de inicializar una constante estática:

```
// !! Error to initialize static constant this way
class StaticVars2
{
    public static const UNIQUESORT:uint;
    function initializeStatic():void
    {
        UNIQUESORT = 16;
    }
}
```

Variables de instancia

Las variables de instancia incluyen propiedades declaradas con las palabras clave `var` y `const`, pero sin la palabra clave `static`. Las variables de este tipo, que se asocian a instancias de clase en lugar de a una clase completa, son útiles para almacenar valores específicos de una instancia. Por ejemplo, la clase Array tiene una propiedad de instancia denominada `length` que almacena el número de elementos de conjunto contenidos en una instancia concreta de la clase Array.

En una subclase no se pueden sustituir las variables de instancia, aunque se declaren como `var` o `const`. Sin embargo, se puede obtener una funcionalidad similar a la sustitución de variables sustituyendo métodos de captador y definidor. Para obtener más información, consulte "Métodos descriptores de acceso (captador y definidor)" en la página 104.

Métodos

Los métodos son funciones que forman parte de una definición de clase. Cuando se crea una instancia de la clase, se vincula un método a esa instancia. A diferencia de una función declarada fuera de una clase, un método sólo puede utilizarse desde la instancia a la que está asociado.

Los métodos se definen con la palabra clave `function`. Al igual que sucede con cualquier propiedad de clase, puede aplicar cualquiera de sus atributos a los métodos, incluyendo private, protected, public, internal, static o un espacio de nombres personalizado. Puede utilizar una sentencia de función como la siguiente:

```
public function sampleFunction():String {}
```

También se puede utilizar una variable a la que se asigna una expresión de función, de la manera siguiente:

```
public var sampleFunction:Function = function () {}
```

En la mayoría de los casos se deseará utilizar una sentencia de función en lugar de una expresión de función por las siguientes razones:

- Las sentencias de función son más concisas y fáciles de leer.
- Permiten utilizar las palabras clave `override` y `final`. Para obtener más información, consulte "Sustitución de métodos" en la página 115.
- Crean un vínculo más fuerte entre el identificador (es decir, el nombre de la función) y el código del cuerpo del método. Como es posible cambiar el valor de una variable con una sentencia de asignación, la conexión entre una variable y su expresión de función se puede hacer más fuerte en cualquier momento. Aunque se puede solucionar este problema declarando la variable con `const` en lugar de `var`, esta técnica no se considera una práctica recomendable, ya que hace que el código sea difícil de leer e impide el uso de las palabras clave `override` y `final`.

Un caso en el que hay que utilizar una expresión de función es cuando se elige asociar una función al objeto prototipo. Para obtener más información, consulte "El objeto prototype" en la página 123.

Métodos constructores

Los métodos constructores, que a veces se llaman simplemente *constructores*, son funciones que comparten el nombre con la clase en la que se definen. Todo el código que se incluya en un método constructor se ejecutará siempre que una instancia de la clase se cree con la palabra clave `new`. Por ejemplo, el código siguiente define una clase simple denominada Example que contiene una sola propiedad denominada `status`. El valor inicial de la variable `status` se establece en la función constructora.

```
class Example
{
    public var status:String;
    public function Example()
    {
        status = "initialized";
    }
}

var myExample:Example = new Example();
trace(myExample.status); // output: initialized
```

Los métodos constructores sólo pueden ser públicos, pero el uso del atributo `public` es opcional. No se puede utilizar en un constructor ninguno de los otros especificadores de control de acceso, incluidos `private`, `protected` e `internal` . Tampoco se puede utilizar un espacio de nombres definido por el usuario con un método constructor.

Un constructor puede hacer una llamada explícita al constructor de su superclase directa utilizando la sentencia`super()`. Si no se llama explícitamente al constructor de la superclase, el compilador inserta automáticamente una llamada antes de la primera sentencia en el cuerpo del constructor. También se puede llamar a métodos de la superclase mediante el prefijo `super` como una referencia a la superclase. Si se decide utilizar `super()` y `super` en el mismo cuerpo de constructor, hay que asegurarse de llamar primero a `super()`. De lo contrario, la referencia `super` no se comportará de la manera esperada. También se debe llamar al constructor `super()` antes que a cualquier sentencia `throw` o `return`.

En el ejemplo siguiente se ilustra lo que sucede si se intenta utilizar la referencia `super` antes de llamar al constructor `super()`. Una nueva clase, ExampleEx, amplía la clase Example. El constructor de ExampleEx intenta acceder a la variable de estado definida en su superclase, pero lo hace antes de llamar a `super()`. La sentencia `trace()` del constructor de ExampleEx produce el valor `null` porque la variable `status` no está disponible hasta que se ejecuta el constructor `super()`.

```
class ExampleEx extends Example
{
    public function ExampleEx()
    {
        trace(super.status);
        super();
    }
}

var mySample:ExampleEx = new ExampleEx(); // output: null
```

Aunque se puede utilizar la sentencia `return` dentro de un constructor, no se permite devolver un valor. Es decir, las sentencias `return` no deben tener expresiones o valores asociados. Por consiguiente, no se permite que los métodos constructores devuelvan valores, lo que significa que no se puede especificar ningún tipo de devolución.

Si no se define un método constructor en la clase, el compilador creará automáticamente un constructor vacío. Si la clase amplía otra clase, el compilador incluirá una llamada `super()` en el constructor que genera.

Métodos estáticos

Los métodos estáticos, también denominados *métodos de clase*, son métodos que se declaran con la palabra clave `static`. Estos métodos, que se asocian a una clase en lugar de a una instancia de clase, son útiles para encapsular la funcionalidad que afecta a algo más que el estado de una instancia individual. Como los métodos estáticos se asocian a una clase como un todo, sólo se puede acceder a dichos métodos a través de una clase, no a través de una instancia de la clase.

Los métodos estáticos son útiles para encapsular la funcionalidad que no se limita a afectar al estado de las instancias de clase. Es decir, un método debe ser estático si proporciona funcionalidad que no afecta directamente al valor de una instancia de clase. Por ejemplo, la clase Date tiene un método estático denominado `parse()`, que convierte una cadena en un número. El método es estático porque no afecta a una instancia individual de la clase. El método `parse()` recibe una cadena que representa un valor de fecha, analiza la cadena y devuelve un número con un formato compatible con la representación interna de un objeto Date. Este método no es un método de instancia porque no tiene sentido aplicar el método a una instancia de la clase Date.

El método `parse()` estático puede compararse con uno de los métodos de instancia de la clase Date, como `getMonth()`. El método `getMonth()` es un método de instancia porque opera directamente en el valor de una instancia recuperando un componente específico, el mes, de una instancia de Date.

Como los métodos estáticos no están vinculados a instancias individuales, no se pueden utilizar las palabras clave `this` o `super` en el cuerpo de un método estático. Las referencias `this` y `super` sólo tienen sentido en el contexto de un método de instancia.

En contraste con otros lenguajes de programación basados en clases, en ActionScript 3.0 los métodos estáticos no se heredan. Para obtener más información, consulte "Propiedades estáticas no heredadas" en la página 117.

Métodos de instancia

Los métodos de instancia son métodos que se declaran sin la palabra clave `static`. Estos métodos, que se asocian a instancias de una clase en lugar de a la clase como un todo, son útiles para implementar funcionalidad que afecta a instancias individuales de una clase. Por ejemplo, la clase Array contiene un método de instancia denominado `sort()`, que opera directamente en instancias de Array.

En el cuerpo de un método de instancia, las variables estáticas y de instancia están dentro del ámbito, lo que significa que se puede hacer referencia a las variables definidas en la misma clase mediante un identificador simple. Por ejemplo, la clase siguiente, CustomArray, amplía la clase Array. La clase CustomArray define una variable estática denominada `arrayCountTotal` para hacer un seguimiento del número total de instancias de clase, una variable de instancia denominada `arrayNumber` que hace un seguimiento del orden en que se crearon las instancias y un método de instancia denominado `getPosition()` que devuelve los valores de estas variables.

```
public class CustomArray extends Array
{
    public static var arrayCountTotal:int = 0;
    public var arrayNumber:int;

    public function CustomArray()
    {
        arrayNumber = ++arrayCountTotal;
    }

    public function getArrayPosition():String
    {
         return ("Array " + arrayNumber + " of " + arrayCountTotal);
    }
}
```

Aunque el código externo a la clase debe hacer referencia a la variable estática `arrayCountTotal` a través del objeto de clase mediante `CustomArray.arrayCountTotal`, el código que reside dentro del cuerpo del método `getPosition()` puede hacer referencia directamente a la variable estática `arrayCountTotal`. Esto se cumple incluso para variables estáticas de superclases. Aunque en ActionScript 3.0 las propiedades estáticas no se heredan, las propiedades estáticas de las superclases están dentro del ámbito. Por ejemplo, la clase Array tiene unas pocas variables estáticas, una de las cuales es una constante denominada `DESCENDING`. El código que reside en una subclase de Array puede hacer referencia a la constante estática `DESCENDING` mediante un identificador simple:

```
public class CustomArray extends Array
{
    public function testStatic():void
    {
        trace(DESCENDING); // output: 2
    }
}
```

El valor de la referencia `this` en el cuerpo de un método de instancia es una referencia a la instancia a la que está asociado el método. El código siguiente muestra que la referencia `this` señala a la instancia que contiene el método:

```
class ThisTest
{
    function thisValue():ThisTest
    {
        return this;
    }
}

var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // output: true
```

La herencia de los métodos de instancia se puede controlar con las palabras clave `override` y `final`. Se puede utilizar el atributo `override` para redefinir un método heredado y el atributo `final` para evitar que las subclases sustituyan un método. Para obtener más información, consulte "Sustitución de métodos" en la página 115.

Métodos descriptores de acceso (captador y definidor)

Las funciones descriptoras de acceso get y set, también denominadas *captadores* y *definidores*, permiten implementar los principios de programación relacionados con la ocultación de información y encapsulación a la vez que ofrecen una interfaz de programación fácil de usar para las clases que se crean. Estas funciones permiten mantener las propiedades de clase como privadas de la clase ofreciendo a los usuarios de la clase acceso a esas propiedades como si accedieran a una variable de clase en lugar de llamar a un método de clase.

La ventaja de este enfoque es que permite evitar las funciones descriptoras de acceso tradicionales con nombres poco flexibles, como `getPropertyName()` y `setPropertyName()`. Otra ventaja de los captadores y definidores es que permiten evitar tener dos funciones públicas por cada propiedad que permita un acceso de lectura y escritura.

La siguiente clase de ejemplo, denominada GetSet, incluye funciones descriptoras de acceso get y set denominadas `publicAccess()` que proporcionan acceso a la variable privada denominada `privateProperty`:

```
class GetSet
{
    private var privateProperty:String;

    public function get publicAccess():String
    {
        return privateProperty;
    }

    public function set publicAccess(setValue:String):void
    {
        privateProperty = setValue;
    }
}
```

Si se intenta directamente acceder a la propiedad `privateProperty` se producirá un error, como se muestra a continuación:

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); // error occurs
```

En su lugar, un usuario de la clase GetSet utilizará algo que parece ser una propiedad denominada `publicAccess`, pero que en realidad es un par de funciones descriptoras de acceso get y set que operan en la propiedad privada denominada `privateProperty`. En el ejemplo siguiente se crea una instancia de la clase GetSet y después se establece el valor de `privateProperty` mediante el descriptor de acceso público denominado `publicAccess`:

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // output: null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // output: hello
```

Las funciones captadoras y definidoras también permiten sustituir propiedades heredadas de una superclase, algo que no es posible al usar variables miembro de clase normales. Las variables miembro de clase que se declaran mediante la palabra clave `var` no se pueden sustituir en una subclase. Sin embargo, las propiedades que se crean mediante funciones captadoras y definidoras no tienen esta restricción. Se puede utilizar el atributo `override` en funciones captadoras y definidoras heredadas de una superclase.

Métodos vinculados

Un método vinculado, a veces denominado *cierre de método*, es simplemente un método que se extrae de su instancia. Los métodos que se pasan como argumentos a una función o se devuelven como valores desde una función son ejemplos de métodos vinculados. Una de las novedades de ActionScript 3.0 es que un método vinculado es similar a un cierre de función, ya que conserva su entorno léxico incluso cuando se extrae de su instancia. Sin embargo, la diferencia clave entre un método vinculado y un cierre de función es que la referencia `this` para un método vinculado permanece vinculada a la instancia que implementa el método. Es decir, la referencia `this` de un método vinculado siempre señala al objeto original que implementó el método. Para los cierres de función, la referencia `this` es genérica, lo que significa que señala al objeto con el que esté relacionada la función cuando se invoque.

Es importante comprender los métodos vinculados para utilizar la palabra clave `this`. Debe recordarse que la palabra clave `this` proporciona una referencia al objeto principal de un método. La mayoría de los programadores que utilizan ActionScript esperan que la palabra clave `this` siempre haga referencia al objeto o la clase que contiene la definición de un método. Sin embargo, sin la vinculación de métodos esto no se cumplirá siempre. Por ejemplo, en las versiones anteriores de ActionScript, la referencia `this` no siempre hacía referencia a la instancia que implementaba el método. En ActionScript 2.0, cuando se extraen métodos de una instancia no sólo no se vincula la referencia `this` a la instancia original, sino que además las variables y los métodos miembro de la clase de la instancia no están disponibles. Esto es no un problema en ActionScript 3.0 porque se crean métodos vinculados automáticamente cuando se pasa un método como parámetro. Los métodos vinculados garantizan que la palabra clave `this` siempre haga referencia al objeto o la clase en que se define un método.

El código siguiente define una clase denominada ThisTest, que contiene un método denominado `foo()` que define el método vinculado y un método denominado `bar()` que devuelve el método vinculado. El código externo a la clase crea una instancia de la clase ThisTest, llama al método `bar()` y almacena el valor devuelto en una variable denominada `myFunc`.

```
class ThisTest
{
    private var num:Number = 3;
    function foo():void // bound method defined
    {
        trace("foo's this: " + this);
        trace("num: " + num);
    }
    function bar():Function
    {
        return foo; // bound method returned
    }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // output: [object global]
myFunc();
/* output:
foo's this: [object ThisTest]
output: num: 3 */
```

Las dos últimas líneas de código muestran que la referencia `this` del método vinculado `foo()` sigue señalando a una instancia de la clase ThisTest, aunque la referencia `this` de la línea inmediatamente anterior señala al objeto global. Además, el método vinculado almacenado en la variable `myFunc` sigue teniendo acceso a las variables miembro de la clase ThisTest. Si se ejecutara este mismo código en ActionScript 2.0, las referencias `this` coincidirían y el valor de la variable `num` sería `undefined`.

La adición de métodos vinculados se aprecia mejor en aspectos como los controladores de eventos, ya que el método `addEventListener()` requiere que se pase una función o un método como un argumento. Para obtener más información, consulte la función Listener definida como método de clase en "Detectores de eventos" en la página 264.

Enumeraciones con clases

Las *enumeraciones* son tipos de datos personalizados que se crean para encapsular un pequeño conjunto de valores. ActionScript 3.0 no ofrece una capacidad de enumeración específica, a diferencia de C++, que incluye la palabra clave `enum`, o Java, con su interfaz Enumeration. No obstante, se pueden crear enumeraciones utilizando clases y constantes estáticas. Por ejemplo, la clase PrintJob en ActionScript 3.0 utiliza una enumeración denominada PrintJobOrientation para almacenar el conjunto de valores formado por `"landscape"` y `"portrait"`, como se muestra en el código siguiente:

```
public final class PrintJobOrientation
{
    public static const LANDSCAPE:String = "landscape";
    public static const PORTRAIT:String = "portrait";
}
```

Por convención, una clase de enumeración se declara con el atributo `final` porque no es necesario ampliarla. La clase sólo contiene miembros estáticos, lo que significa que no se crean instancias de la clase, sino que se accede a los valores de la enumeración directamente a través del objeto de clase, como se indica en el siguiente fragmento de código:

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
    if (pj.orientation == PrintJobOrientation.PORTRAIT)
    {
        ...
    }
    ...
}
```

Todas las clases de enumeración en ActionScript 3.0 sólo contienen variables de tipo String, int o uint. La ventaja de utilizar enumeraciones en lugar de valores literales numéricos o de cadena es que es más fácil detectar errores tipográficos con las enumeraciones. Si se escribe incorrectamente el nombre de una enumeración, el compilador de ActionScript genera un error. Si se utilizan valores literales, el compilador no mostrará ninguna advertencia en caso de que encuentre una palabra escrita incorrectamente o se utilice un número incorrecto. En el ejemplo anterior, el compilador genera un error si el nombre de la constante de enumeración es incorrecto, como se indica en el siguiente fragmento:

```
    if (pj.orientation == PrintJobOrientation.PORTRAI) // compiler error
```

Sin embargo, el compilador no generará un error si se escribe incorrectamente un valor literal de cadena, como se muestra a continuación:

```
    if (pj.orientation == "portrai") // no compiler error
```

Otra técnica para crear enumeraciones también implica crear una clase independiente con propiedades estáticas para la enumeración. No obstante, esta técnica difiere en que cada una de las propiedades estáticas contiene una instancia de la clase en lugar de una cadena o un valor entero. Por ejemplo, el código siguiente crea una clase de enumeración para los días de la semana:

```
public final class Day
{
    public static const MONDAY:Day = new Day();
    public static const TUESDAY:Day = new Day();
    public static const WEDNESDAY:Day = new Day();
    public static const THURSDAY:Day = new Day();
    public static const FRIDAY:Day = new Day();
    public static const SATURDAY:Day = new Day();
    public static const SUNDAY:Day = new Day();
}
```

En ActionScript 3.0 no se emplea esta técnica, pero la utilizan muchos desarrolladores que prefieren la verificación de tipos mejorada que proporciona. Por ejemplo, un método que devuelve un valor de enumeración puede restringir el valor devuelto al tipo de datos de la enumeración. El código siguiente muestra, además de una función que devuelve un día de la semana, una llamada a función que utiliza el tipo de la enumeración como una anotación de tipo:

```
function getDay():Day
{
    var date:Date = new Date();
    var retDay:Day;
    switch (date.day)
    {
        case 0:
            retDay = Day.MONDAY;
            break;
        case 1:
            retDay = Day.TUESDAY;
            break;
        case 2:
            retDay = Day.WEDNESDAY;
            break;
        case 3:
            retDay = Day.THURSDAY;
            break;
        case 4:
            retDay = Day.FRIDAY;
            break;
        case 5:
            retDay = Day.SATURDAY;
            break;
        case 6:
            retDay = Day.SUNDAY;
            break;
    }
    return retDay;
}

var dayOfWeek:Day = getDay();
```

También se puede mejorar la clase Day de forma que asocie un entero con cada día de la semana y proporcione un método `toString()` que devuelva una representación de cadena del día. Se puede mejorar la clase Day de esta manera como ejercicio.

Clases de activos incorporados

ActionScript 3.0 utiliza clases especiales, denominadas *clases de activos incorporados*, para representar elementos incorporados. Un *activo incorporado* es un activo, como un sonido, una imagen o una fuente, que se incluye en un archivo SWF en tiempo de compilación. Incorporar un activo en lugar de cargarlo dinámicamente garantiza que estará disponible en tiempo de ejecución, pero a cambio el tamaño del archivo SWF será mayor.

Utilización de clases de activos incorporados en Flash

Para incorporar un activo, hay que añadir primero el activo a una biblioteca de archivo FLA. A continuación, hay que usar la propiedad linkage del activo para asignar un nombre a la clase de activo incorporado del activo. Si no se encuentra una clase con ese nombre en la ruta de clases, se generará una clase automáticamente. Después se puede utilizar una instancia de la clase de activo incorporado y utilizar las propiedades y los métodos definidos por la clase. Por ejemplo, se puede utilizar el código siguiente para reproducir un sonido incorporado vinculado a una clase de activo incorporado denominada PianoMusic:

```
var piano:PianoMusic = new PianoMusic();
var sndChannel:SoundChannel = piano.play();
```

Interfaces

Una interfaz es una colección de declaraciones de métodos que permite la comunicación entre objetos que no están relacionados. Por ejemplo, ActionScript 3.0 define la interfaz IEventDispatcher, que contiene declaraciones de métodos que una clase puede utilizar para controlar objetos de evento. La interfaz IEventDispatcher establece una forma estándar de pasar objetos de evento entre objetos. El código siguiente muestra la definición de la interfaz IEventDispatcher:

```
public interface IEventDispatcher
{
    function addEventListener(type:String, listener:Function,
            useCapture:Boolean=false, priority:int=0,
            useWeakReference:Boolean = false):void;
    function removeEventListener(type:String, listener:Function,
            useCapture:Boolean=false):void;
    function dispatchEvent(event:Event):Boolean;
    function hasEventListener(type:String):Boolean;
    function willTrigger(type:String):Boolean;
}
```

Las interfaces se basan en la distinción entre la interfaz de un método y su implementación. La interfaz de un método incluye toda la información necesaria para invocar dicho método, como el nombre del método, todos sus parámetros y el tipo que devuelve. La implementación de un método no sólo incluye la información de la interfaz, sino también las sentencias ejecutables que implementan el comportamiento del método. Una definición de interfaz sólo contiene interfaces de métodos; cualquier clase que implemente la interfaz debe encargarse de definir las implementaciones de los métodos.

En ActionScript 3.0, la clase EventDispatcher implementa la interfaz IEventDispatcher definiendo todos los métodos de la interfaz IEventDispatcher y añadiendo el código a cada uno de los métodos. El código siguiente es un fragmento de la definición de la clase EventDispatcher:

```
public class EventDispatcher implements IEventDispatcher
{
    function dispatchEvent(event:Event):Boolean
    {
        /* implementation statements */
    }

    ...
}
```

La interfaz IEventDispatcher constituye un protocolo que las instancias de EventDispatcher utilizan para procesar objetos de evento y pasárselos a otros objetos que también tienen implementada la interfaz IEventDispatcher.

Otra forma de describir una interfaz es decir que define un tipo de datos de la misma manera que una clase. Por consiguiente, una interfaz se puede utilizar como una anotación de tipo, igual que una clase. Al ser un tipo de datos, una interfaz también se puede utilizar con operadores, como los operadores `is` y `as`, que requieren un tipo de datos. Sin embargo, a diferencia de una clase, no se puede crear una instancia de una interfaz. Esta distinción hace que muchos programadores consideren que las interfaces son tipos de datos abstractos y las clases son tipos de datos concretos.

Definición de una interfaz

La estructura de una definición de interfaz es similar a la de una definición de clase, con la diferencia de que una interfaz sólo puede contener métodos sin código de método. Las interfaces no pueden incluir variables ni constantes, pero pueden incluir captadores y definidores. Para definir una interfaz se utiliza la palabra clave `interface`. Por ejemplo, la siguiente interfaz, IExternalizable, forma parte del paquete flash.utils en ActionScript 3.0. La interfaz IExternalizable define un protocolo para serializar un objeto, lo que implica convertir un objeto en un formato adecuado para el almacenamiento en un dispositivo o para el transporte a través de la red.

```
public interface IExternalizable
{
    function writeExternal(output:IDataOutput):void;
    function readExternal(input:IDataInput):void;
}
```

Hay que tener en cuenta que la interfaz IExternalizable se declara con el modificador de control de acceso `public`. Las definiciones de interfaz sólo pueden modificarse mediante los especificadores de control de acceso `public` e `internal`. Las declaraciones de métodos en una definición de interfaz no pueden incluir ningún especificador de control de acceso.

ActionScript 3.0 sigue una convención por la que los nombres de interfaz empiezan por una `I` mayúscula, pero se puede utilizar cualquier identificador válido como nombre de interfaz. Las definiciones de interfaz se suelen colocar en el nivel superior de un paquete. No pueden colocarse en una definición de clase ni en otra definición de interfaz.

Las interfaces pueden ampliar una o más interfaces. Por ejemplo, la siguiente interfaz, IExample, amplía la interfaz IExternalizable:

```
public interface IExample extends IExternalizable
{
    function extra():void;
}
```

Cualquier clase que implemente la interfaz IExample debe incluir implementaciones no sólo para el método `extra()`, sino también para los métodos `writeExternal()` y `readExternal()` heredados de la interfaz IExternalizable.

Implementación de una interfaz en una clase

Una clase es el único elemento del lenguaje ActionScript 3.0 que puede implementar una interfaz. Se puede utilizar la palabra clave `implements` en una declaración de clase para implementar una o más interfaces. En el ejemplo siguiente se definen dos interfaces, IAlpha e IBeta, y una clase, Alpha, que implementa las dos interfaces:

```
interface IAlpha
{
    function foo(str:String):String;
}

interface IBeta
{
    function bar():void;
}

class Alpha implements IAlpha, IBeta
{
    public function foo(param:String):String {}
    public function bar():void {}
}
```

En una clase que implementa una interfaz, los métodos implementados deben:

- Utilizar el identificador de control de acceso `public`.
- Utilizar el mismo nombre que el método de interfaz.
- Tener el mismo número de parámetros, cada uno con un tipo de datos que coincida con los tipos de datos de los parámetros del método de interfaz.
- Devolver el mismo tipo de datos.

```
public function foo(param:String):String {}
```

Sin embargo, hay cierta flexibilidad para asignar nombres a los parámetros de métodos implementados. Aunque el número de parámetros y el tipo de datos de cada parámetro del método implementado deben coincidir con los del método de interfaz, los nombres de los parámetros no tienen que coincidir. Por ejemplo, en el ejemplo anterior el parámetro del método `Alpha.foo()` se denomina `param`:

En el método de interfaz `IAlpha.foo()`, el parámetro se denomina `str`:

```
function foo(str:String):String;
```

También hay cierta flexibilidad con los valores predeterminados de parámetros. Una definición de interfaz puede incluir declaraciones de funciones con valores predeterminados de parámetros. Un método que implementa una declaración de función de este tipo debe tener un valor predeterminado de parámetro que sea miembro del mismo tipo de datos que el valor especificado en la definición de interfaz, pero el valor real no tiene que coincidir. Por ejemplo, el código siguiente define una interfaz que contiene un método con un valor predeterminado de parámetro igual a 3:

```
interface IGamma
{
    function doSomething(param:int = 3):void;
}
```

La siguiente definición de clase implementa la interfaz Igamma, pero utiliza un valor predeterminado de parámetro distinto:

```
class Gamma implements IGamma
{
    public function doSomething(param:int = 4):void {}
}
```

La razón de esta flexibilidad es que las reglas para implementar una interfaz se han diseñado específicamente para garantizar la compatibilidad de los tipos de datos y no es necesario exigir que los nombres de parámetros y los valores predeterminados de los parámetros sean idénticos para alcanzar ese objetivo.

Herencia

La herencia es una forma de reutilización de código que permite a los programadores desarrollar clases nuevas basadas en clases existentes. Las clases existentes se suelen denominar *clases base* o *superclases*, y las clases nuevas se denominan *subclases*. Una ventaja clave de la herencia es que permite reutilizar código de una clase base manteniendo intacto el código existente. Además, la herencia no requiere realizar ningún cambio en la interacción entre otras clases y la clase base. En lugar de modificar una clase existente que puede haber sido probada minuciosamente o que ya se está utilizando, la herencia permite tratar esa clase como un módulo integrado que se puede ampliar con propiedades o métodos adicionales. Se utiliza la palabra clave `extends` para indicar que una clase hereda de otra clase.

La herencia también permite beneficiarse del *polimorfismo* en el código. El polimorfismo es la capacidad de utilizar un solo nombre de método para un método que se comporta de distinta manera cuando se aplica a distintos tipos de datos. Un ejemplo sencillo es una clase base denominada Shape con dos subclases denominadas Circle y Square. La clase Shape define un método denominado `area()` que devuelve el área de la forma. Si se implementa el polimorfismo, se puede llamar al método `area()` en objetos de tipo Circle y Square, y se harán automáticamente los cálculos correctos. La herencia activa el polimorfismo al permitir que las subclases hereden y redefinan (*sustituyan*) los métodos de la clase base. En el siguiente ejemplo, se redefine el método `area()` mediante las clases Circle y Square:

```
class Shape
{
    public function area():Number
    {
        return NaN;
    }
}

class Circle extends Shape
{
    private var radius:Number = 1;
    override public function area():Number
    {
        return (Math.PI * (radius * radius));
    }
}

class Square extends Shape
{
    private var side:Number = 1;
    override public function area():Number
    {
        return (side * side);
    }
}

var cir:Circle = new Circle();
trace(cir.area()); // output: 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // output: 1
```

Como cada clase define un tipo de datos, el uso de la herencia crea una relación especial entre una clase base y una clase que la amplía. Una subclase posee todas las propiedades de su clase base, lo que significa que siempre se puede sustituir una instancia de una subclase como una instancia de la clase base. Por ejemplo, si un método define un parámetro de tipo Shape, se puede pasar un argumento de tipo Circle, ya que Circle amplía Shape, como se indica a continuación:

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

Herencia y propiedades de instancia

Todas las subclases heredan una propiedad de instancia definida con la palabra clave `function`, `var` o `const`, con tal de que no se haya declarado la propiedad con el atributo `private` en la clase base. Por ejemplo, la clase Event en ActionScript 3.0 tiene diversas subclases que heredan propiedades comunes a todos los objetos de evento.

Para algunos tipos de eventos, la clase Event contiene todas las propiedades necesarias para definir el evento. Estos tipos de eventos no requieren más propiedades de instancia que las definidas en la clase Event. Algunos ejemplos de estos eventos son el evento `complete`, que se produce cuando los datos se han cargado correctamente, y el evento `connect`, que se produce cuando se establece una conexión de red.

El ejemplo siguiente es un fragmento de la clase Event que muestra algunas de las propiedades y los métodos que heredarán las subclases. Como las propiedades se heredan, una instancia de cualquier subclase puede acceder a estas propiedades.

```
public class Event
{
    public function get type():String;
    public function get bubbles():Boolean;
    ...

    public function stopPropagation():void {}
    public function stopImmediatePropagation():void {}
    public function preventDefault():void {}
    public function isDefaultPrevented():Boolean {}
    ...
}
```

Otros tipos de eventos requieren propiedades únicas que no están disponibles en la clase Event. Estos eventos se definen creando subclases de la clase Event para añadir nuevas propiedades a las propiedades definidas en la clase Event. Un ejemplo de una subclase de este tipo es la clase MouseEvent, que añade propiedades únicas a eventos asociados con el movimiento o los clics del ratón, como los eventos `mouseMove` y `click`. El ejemplo siguiente es un fragmento de la clase MouseEvent que muestra la definición de propiedades que se han añadido a la subclase y, por tanto, no existen en la clase base:

```
public class MouseEvent extends Event
{
    public static const CLICK:String= "click";
    public static const MOUSE_MOVE:String = "mouseMove";
    ...

    public function get stageX():Number {}
    public function get stageY():Number {}
    ...
}
```

Herencia y especificadores de control de acceso

Si una propiedad se declara con la palabra clave `public`, estará visible en cualquier parte del código. Esto significa que la palabra clave `public`, a diferencia de las palabras clave `private`, `protected` e `internal`, no restringe de ningún modo la herencia de propiedades.

Si se declara una propiedad con la palabra clave `private`, sólo será visible en la clase que la define, lo que significa que ninguna subclase la heredará. Este comportamiento es distinto del de las versiones anteriores de ActionScript, en las que el comportamiento de la palabra clave `private` era más parecido al de la palabra clave `protected` de ActionScript 3.0.

La palabra clave `protected` indica que una propiedad es visible en la clase que la define y en todas sus subclases. A diferencia de la palabra clave `protected` del lenguaje de programación Java, la palabra clave `protected` de ActionScript 3.0 no hace que una propiedad esté visible para todas las clases de un mismo paquete. En ActionScript 3.0 sólo las subclases pueden acceder a una propiedad declarada con la palabra clave `protected`. Además, una propiedad protegida estará visible para una subclase tanto si la subclase está en el mismo paquete que la clase base como si está en un paquete distinto.

Para limitar la visibilidad de una propiedad al paquete en el que se define, se debe utilizar la palabra clave `internal` o no utilizar ningún especificador de control de acceso. Cuando no se especifica ninguno especificador de control de acceso, el predeterminado es `internal`. Una propiedad marcada como `internal` sólo podrá ser heredada por una subclase que resida en el mismo paquete.

Se puede utilizar el ejemplo siguiente para ver cómo afecta cada uno de los especificadores de control de acceso a la herencia más allá de los límites del paquete. El código siguiente define una clase principal de aplicación denominada AccessControl y otras dos clases denominadas Base y Extender. La clase Base está en un paquete denominado foo y la clase Extender, que es una subclase de la clase Base, está en un paquete denominado bar. La clase AccessControl sólo importa la clase Extender y crea una instancia de Extender que intenta acceder a una variable denominada `str` definida en la clase Base. La variable `str` se declara como `public` de forma que el código se compile y ejecute como se indica en el siguiente fragmento:

```
// Base.as in a folder named foo
package foo
{
    public class Base
    {
        public var str:String = "hello"; // change public on this line
    }
}

// Extender.as in a folder named bar
package bar
{
    import foo.Base;
    public class Extender extends Base
    {
        public function getString():String {
            return str;
        }
    }
}

// main application class in file named AccessControl.as
package
{
    import flash.display.MovieClip;
    import bar.Extender;
    public class AccessControl extends MovieClip
    {
        public function AccessControl()
        {
            var myExt:Extender = new Extender();
            trace(myExt.str);// error if str is not public
            trace(myExt.getString()); // error if str is private or internal
        }
    }
}
```

Para ver cómo afectan los otros especificadores de control de acceso a la compilación y la ejecución del ejemplo anterior, se debe cambiar el especificador de control de acceso de la variable `str` a `private`, `protected` o `internal` después de eliminar o marcar como comentario la línea siguiente de la clase `AccessControl`:

```
trace(myExt.str);// error if str is not public
```

No se permite la sustitución de variables

Las propiedades declaradas con la palabra clave `var` o `const` se pueden heredar, pero no se pueden sustituir. Para sustituir una propiedad hay que redefinirla en una subclase. El único tipo de propiedad que se puede sustituir son los métodos (es decir, propiedades declaradas con la palabra clave `function`). Aunque no es posible sustituir una variable de instancia, se puede obtener una funcionalidad similar creando métodos captador y definidor para la variable de instancia y sustituyendo los métodos. Para obtener más información, consulte "Sustitución de métodos" en la página 115.

Sustitución de métodos

Para sustituir un método hay que redefinir el comportamiento de un método heredado. Los métodos estáticos no se heredan y no se pueden sustituir. Sin embargo, las subclases heredan los métodos de instancia, que se pueden sustituir con tal de que se cumplan los dos criterios siguientes:

- El método de instancia no se ha declarado con la palabra clave `final` en la clase base. Si se utiliza la palabra clave `final` con un método de instancia, indica la intención del programador de evitar que las subclases sustituyan el método.
- El método de instancia no se declara con el especificador de control de acceso `private` de la clase base. Si se marca un método como `private` en la clase base, no hay que usar la palabra clave `override` al definir un método con el mismo nombre en la subclase porque el método de la clase base no será visible para la subclase.

  Para sustituir un método de instancia que cumpla estos criterios, la definición del método en la subclase debe utilizar la palabra clave `override` y debe coincidir con la versión de la superclase del método en los siguientes aspectos:

- El método sustituto debe tener el mismo nivel de control de acceso que el método de la clase base. Los métodos marcados como internal tienen el mismo nivel de control de acceso que los métodos que no tienen especificador de control de acceso.
- El método sustituto debe tener el mismo número de parámetros que el método de la clase base.
- Los parámetros del método sustituto deben tener las mismas anotaciones de tipo de datos que los parámetros del método de la clase base.
- El método sustituto debe devolver el mismo tipo de datos que el método de la clase base.

Sin embargo, los nombres de los parámetros del método sustituto no tienen que coincidir con los nombres de los parámetros de la clase base, con tal de que el número de parámetros y el tipo de datos de cada parámetro coincidan.

La sentencia super

Al sustituir un método, los programadores generalmente desean añadir funcionalidad al comportamiento del método de la superclase que van a sustituir, en lugar de sustituir completamente el comportamiento. Esto requiere un mecanismo que permita a un método de una subclase llamar a la versión de sí mismo de la superclase. La sentencia `super` proporciona este mecanismo, ya que contiene una referencia a la superclase inmediata. El ejemplo siguiente define una clase denominada Base que contiene un método denominado `thanks()` y una subclase de la clase Base denominada Extender que reemplaza el método `thanks()`. El método `Extender.thanks()` utiliza la sentencia `super` para llamar a `Base.thanks()`.

```
package {
    import flash.display.MovieClip;
    public class SuperExample extends MovieClip
    {
        public function SuperExample()
        {
            var myExt:Extender = new Extender()
            trace(myExt.thanks()); // output: Mahalo nui loa
        }
    }
}

class Base {
    public function thanks():String
    {
        return "Mahalo";
    }
}

class Extender extends Base
{
    override public function thanks():String
    {
        return super.thanks() + " nui loa";
    }
}
```

Sustitución de captadores y definidores

Aunque las variables definidas en una superclase no se pueden sustituir, sí se pueden sustituir los captadores y definidores. Por ejemplo, el código siguiente sustituye un captador denominado `currentLabel` definido en la clase MovieClip en ActionScript 3.0:

```
package
{
    import flash.display.MovieClip;
    public class OverrideExample extends MovieClip
    {
        public function OverrideExample()
        {
            trace(currentLabel)
        }
        override public function get currentLabel():String
        {
            var str:String = "Override: ";
            str += super.currentLabel;
            return str;
        }
    }
}
```

El resultado de la sentencia `trace()` en el constructor de la clase OverrideExample es `Override: null`, lo que indica que el ejemplo pudo sustituir la propiedad heredada `currentLabel`.

Propiedades estáticas no heredadas

Las subclases no heredan las propiedades estáticas. Esto significa que no se puede acceder a las propiedades estáticas a través de una instancia de una subclase. Sólo se puede acceder a una propiedad estática a través del objeto de clase en el que está definida. Por ejemplo, en el código siguiente se define una clase base denominada Base y una subclase que amplía Base denominada Extender. Se define una variable estática denominada `test` en la clase Base. El código del siguiente fragmento no se compila en modo estricto y genera un error en tiempo de ejecución en modo estándar.

```
package {
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// error
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base { }
```

La única manera de acceder a la variable estática `test` es a través del objeto de clase, como se indica en el código siguiente:

```
Base.test;
```

No obstante, se puede definir una propiedad de instancia con el mismo nombre que una propiedad estática. Esta propiedad de instancia puede definirse en la misma clase que la propiedad estática o en una subclase. Por ejemplo, la clase Base del ejemplo anterior podría tener una propiedad de instancia denominada `test`. El código siguiente se compila y ejecuta, ya que la propiedad de instancia es heredada por la clase Extender. El código también se compilará y ejecutará si la definición de la variable de instancia de prueba se mueve (en lugar de copiarse) a la clase Extender.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// output: instance
        }
    }
}

class Base
{
    public static var test:String = "static";
    public var test:String = "instance";
}

class Extender extends Base {}
```

Propiedades estáticas y cadena de ámbitos

Aunque las propiedades estáticas no se heredan, están en la cadena de ámbitos de la clase en la que se definen y de cualquier subclase de dicha clase. Así, se dice que las propiedades estáticas están *dentro del ámbito* de la clase en la que se definen y de sus subclases. Esto significa que una propiedad estática es directamente accesible en el cuerpo de la clase en la que se define y en cualquier subclase de dicha clase.

En el ejemplo siguiente se modifican las clases definidas en el ejemplo anterior para mostrar que la variable estática `test` definida en la clase Base está en el ámbito de la clase Extender. Es decir, la clase Extender puede acceder a la variable estática `test` sin añadir a la variable el nombre de la clase que define `test` como prefijo.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base
{
    public function Extender()
    {
        trace(test); // output: static
    }

}
```

Si se define una propiedad de instancia que utiliza el mismo nombre que una propiedad estática en la misma clase o en una superclase, la propiedad de instancia tiene precedencia superior en la cadena de ámbitos. Se dice que la propiedad de instancia *oculta* la propiedad estática, lo que significa que se utiliza el valor de la propiedad de instancia en lugar del valor de la propiedad estática. Por ejemplo, el código siguiente muestra que si la clase Extender define una variable de instancia denominada `test`, la sentencia `trace()` utiliza el valor de la variable de instancia en lugar del valor de la variable estática:

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
        }
    }
}

class Base
{
    public static var test:String = "static";
}

class Extender extends Base
{
    public var test:String = "instance";
    public function Extender()
    {
        trace(test); // output: instance
    }

}
```

Temas avanzados

Esta sección empieza con una historia abreviada de ActionScript y la programación orientada a objetos, y continúa con una descripción del modelo de objetos de ActionScript 3.0 y de cómo permite que la nueva máquina virtual de ActionScript (AVM2) sea considerablemente más rápida que la de las versiones anteriores de Flash Player, que incluyen la máquina virtual de ActionScript antigua (AVM1).

Historia de la implementación de la programación orientada a objetos en ActionScript

Como el diseño de ActionScript 3.0 se ha basado en las versiones anteriores de ActionScript, puede resultar útil comprender la evolución del modelo de objetos de ActionScript. ActionScript empezó siendo un mecanismo sencillo de creación de scripts para las primeras versiones de la herramienta de edición Flash. Con el tiempo, los programadores empezaron a crear aplicaciones cada vez más complejas con ActionScript. En respuesta a las necesidades de los programadores, en cada nueva versión se añadieron características al lenguaje para facilitar la creación de aplicaciones complejas.

ActionScript 1.0

ActionScript 1.0 es la versión del lenguaje utilizada en Flash Player 6 y en versiones anteriores. En esta fase inicial del desarrollo, el modelo de objetos de ActionScript ya se basaba en el concepto de objeto como tipo de datos básico. Un objeto de ActionScript es un tipo de datos compuesto con un conjunto de *propiedades*. Al describir el modelo de objetos, el término *propiedades* abarca todo lo que está asociado a un objeto, como las variables, las funciones o los métodos.

Aunque esta primera generación de ActionScript no permitía definir clases con una palabra clave `class`, se podía definir una clase mediante un tipo de objeto especial denominado objeto prototipo. En lugar de utilizar una palabra clave `class` para crear una definición de clase abstracta a partir de la cual se puedan crear instancias de objetos, como se hace en otros lenguajes basados en clases como Java y C++, los lenguajes basados en prototipos como ActionScript 1.0 utilizan un objeto existente como modelo (o prototipo) para crear otros objetos. En un lenguaje basado en clases, los objetos pueden señalar a una clase como su plantilla; en cambio, en un lenguaje basado en prototipos los objetos señalan a otro objeto, el prototipo, que es su plantilla.

Para crear una clase en ActionScript 1.0, se define una función constructora para dicha clase. En ActionScript, las funciones son objetos reales, no sólo definiciones abstractas. La función constructora que se crea sirve como objeto prototipo para las instancias de dicha clase. El código siguiente crea una clase denominada Shape y define una propiedad denominada `visible` establecida en `true` de manera predeterminada:

```
// base class
function Shape() {}
// Create a property named visible.
Shape.prototype.visible = true;
```

Esta función constructora define una clase Shape de la que se puede crear una instancia mediante el operador `new`, de la manera siguiente:

```
myShape = new Shape();
```

De la misma manera que el objeto de función constructora de `Shape()` es el prototipo para instancias de la clase Shape, también puede ser el prototipo para subclases de Shape (es decir, otras clases que amplíen la clase Shape).

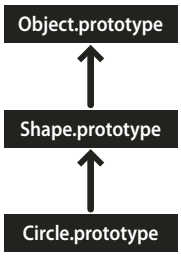
La creación de una clase que es una subclase de la clase Shape es un proceso en dos pasos. En primer lugar debe crearse la clase definiendo una función constructora, de la manera siguiente:

```
// child class
function Circle(id, radius)
{
this.id = id;
this.radius = radius;
}
```

En segundo lugar, se utiliza el operador `new` para declarar que la clase Shape es el prototipo para la clase Circle. De manera predeterminada, cualquier clase que se cree utilizará la clase Object como prototipo, lo que significa que `Circle.prototype` contiene actualmente un objeto genérico (una instancia de la clase Object). Para especificar que el prototipo de Circle es Shape y no Object, se debe utilizar el código siguiente para cambiar el valor de `Circle.prototype` de forma que contenga un objeto Shape en lugar de un objeto genérico:

```
// Make Circle a subclass of Shape.
Circle.prototype = new Shape();
```

La clase Shape y la clase Circle ahora están vinculadas en una relación de herencia que se suele llamar *cadena de prototipos*. El diagrama ilustra las relaciones de una cadena de prototipos:



La clase base al final de cada cadena de prototipos es la clase Object. La clase Object contiene una propiedad estática denominada `Object.prototype` que señala al objeto prototipo base para todos los objetos creados en ActionScript 1.0. El siguiente objeto de la cadena de prototipos de ejemplo es el objeto Shape. Esto se debe a que la propiedad `Shape.prototype` no se ha establecido explícitamente, por lo que sigue conteniendo un objeto genérico (una instancia de la clase Object). El vínculo final de esta cadena es la clase Circle, que está vinculada a su prototipo, la clase Shape (la propiedad `Circle.prototype` contiene un objeto Shape).

Si se crea una instancia de la clase Circle, como en el siguiente ejemplo, la instancia hereda la cadena de prototipos de la clase Circle:

```
// Create an instance of the Circle class.
myCircle = new Circle();
```

Antes se creó una propiedad denominada `visible` como un miembro de la clase Shape. En este ejemplo, la propiedad `visible` no existe como parte del objeto `myCircle`, sólo existe como miembro del objeto Shape; sin embargo, la siguiente línea de código devuelve `true`:

```
trace(myCircle.visible); // output: true
```

Flash Player puede determinar que el objeto `myCircle` hereda la propiedad `visible` recorriendo la cadena de prototipos. Al ejecutar este código, Flash Player busca primero una propiedad denominada `visible` en las propiedades del objeto `myCircle`, pero no la encuentra. A continuación, busca en el objeto `Circle.prototype`, pero sigue sin encontrar una propiedad denominada `visible`. Sigue por la cadena de prototipos hasta que encuentra la propiedad `visible` definida en el objeto `Shape.prototype` y devuelve el valor de dicha propiedad.

En esta sección se omiten muchos de los detalles y las complejidades de la cadena de prototipos por simplificar. El objetivo es proporcionar información suficiente para comprender el modelo de objetos de ActionScript 3.0.

ActionScript 2.0

En ActionScript 2.0 se incluyeron palabras clave nuevas, como `class`, `extends`, `public` y `private`, que permitían definir clases de una manera familiar para cualquiera que trabaje con lenguajes basados en clases como Java y C++. Es importante saber que el mecanismo de herencia subyacente no ha cambiado entre ActionScript 1.0 y ActionScript 2.0. En ActionScript 2.0 simplemente se ha añadido una nueva sintaxis para la definición de clases. La cadena de prototipos funciona de la misma manera en ambas versiones del lenguaje.

La nueva sintaxis introducida en ActionScript 2.0, que se muestra en el siguiente fragmento, permite definir clases de una manera más intuitiva para muchos programadores:

```
// base class
class Shape
{
var visible:Boolean = true;
}
```

En ActionScript 2.0 también se introdujeron las anotaciones de tipos de datos para la verificación de tipos en tiempo de compilación. Esto permite declarar que la propiedad `visible` del ejemplo anterior sólo debe contener un valor booleano. La nueva palabra clave `extends` también simplifica el proceso de crear una subclase. En el siguiente ejemplo, el proceso que requería dos pasos en ActionScript 1.0 se realiza con un solo paso mediante la palabra clave `extends`:

```
// child class
class Circle extends Shape
{
    var id:Number;
    var radius:Number;
    function Circle(id, radius)
    {
        this.id = id;
        this.radius = radius;
        }
}
```

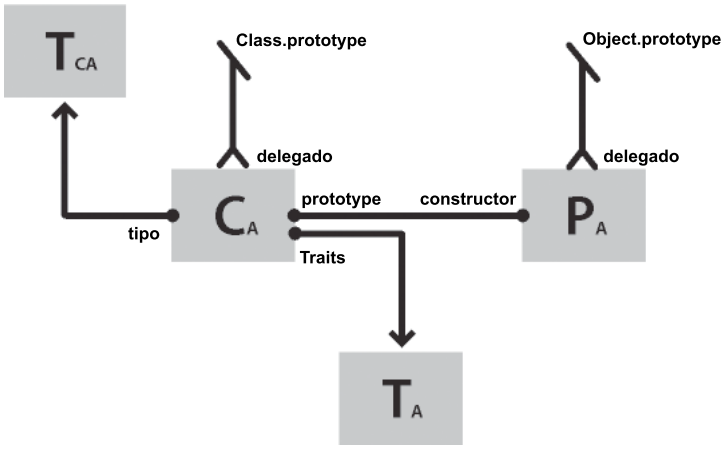
Ahora el constructor se declara como parte de la definición de clase y las propiedades de clase `id` y `radius` también deben declararse explícitamente.

En ActionScript 2.0 también se permite definir interfaces, lo que permite refinar los programas orientados a objetos con protocolos definidos formalmente para la comunicación entre objetos.

El objeto de clase de ActionScript 3.0

Un paradigma común de la programación orientada a objetos, que se suele asociar con Java y C++, utiliza las clases para definir tipos de objetos. Los lenguajes de programación que adoptan este paradigma también tienden a utilizar clases para crear instancias del tipo de datos definido por la clase. ActionScript utiliza clases para ambos propósitos, pero sus raíces como lenguaje basado en prototipos le añaden una característica interesante. ActionScript crea para cada definición de clase un objeto de clase especial que permite compartir el comportamiento y el estado. Sin embargo, para muchos programadores que utilizan ActionScript, esta distinción puede no tener ninguna consecuencia práctica en la programación. ActionScript 3.0 se ha diseñado de forma que se puedan crear sofisticadas aplicaciones orientadas a objetos sin tener que utilizar, ni si quiera comprender, estos objetos de clase especiales. En esta sección se tratan estos temas en profundidad, para los programadores expertos que deseen utilizar objetos de clase.

En el diagrama siguiente se muestra la estructura de un objeto de clase que representa una clase simple denominada A que se define con la sentencia `class A {}`:



Cada rectángulo del diagrama representa un objeto. Cada objeto del diagrama tiene un carácter de subíndice para representar que pertenece a la clase A. El objeto de clase (CA) contiene referencias a una serie de otros objetos importantes. Un objeto traits de instancia (TA) almacena las propiedades de instancia definidas en una definición de clase. Un objeto traits de clase (TCA) representa el tipo interno de la clase y almacena las propiedades estáticas definidas por la clase (el carácter de subíndice C significa "class"). El objeto prototipo (PA) siempre hace referencia al objeto de clase al que se asoció originalmente mediante la propiedad `constructor`.

El objeto traits

El objeto traits, que es una novedad en ActionScript 3.0, se implementó para mejorar el rendimiento. En versiones anteriores de ActionScript, la búsqueda de nombres era un proceso lento, ya que Flash Player recorría la cadena de prototipos. En ActionScript 3.0, la búsqueda de nombres es mucho más rápida y eficaz, ya que las propiedades heredadas se copian desde las superclases al objeto traits de las subclases.

No se puede acceder directamente al objeto traits desde el código, pero las mejoras de rendimiento y de uso de la memoria reflejan el efecto que produce. El objeto traits proporciona a la AVM2 información detallada sobre el diseño y el contenido de una clase. Con este conocimiento, la AVM2 puede reducir considerablemente el tiempo de ejecución, ya que generalmente podrá generar instrucciones directas de código máquina para acceder a las propiedades o llamar a métodos directamente sin tener que realizar una búsqueda lenta de nombres.

Gracias al objeto traits, la huella en la memoria de un objeto puede ser considerablemente menor que la de un objeto similar en las versiones anteriores de ActionScript. Por ejemplo, si una clase está cerrada (es decir, no se declara como dinámica), una instancia de la clase no necesita una tabla hash para propiedades añadidas dinámicamente y puede contener poco más que un puntero a los objetos traits y espacio para las propiedades fijas definidas en la clase. En consecuencia, un objeto que requería 100 bytes de memoria en ActionScript 2.0 puede requerir tan sólo 20 bytes de memoria en ActionScript 3.0.

Nota: *el objeto traits es un detalle de implementación interna y no hay garantías de que no cambie o incluso desaparezca en versiones futuras de ActionScript.*

El objeto prototype

Cada clase de objeto de ActionScript tiene una propiedad denominada `prototype`, que es una referencia al objeto prototipo de la clase. El objeto prototipo es un legado de las raíces de ActionScript como lenguaje basado en prototipos. Para obtener más información, consulte "Historia de la implementación de la programación orientada a objetos en ActionScript" en la página 119.

La propiedad `prototype` es de sólo lectura, lo que significa que no se puede modificar para que señale a otros objetos. Esto ha cambiado con respecto a la propiedad `prototype` de una clase en las versiones anteriores de ActionScript, en las que se podía reasignar el prototipo de forma que señalara a una clase distinta. Aunque la propiedad `prototype` es de sólo lectura, el objeto prototipo al que hace referencia no lo es. Es decir, se pueden añadir propiedades nuevas al objeto prototipo. Las propiedades añadidas al objeto prototipo son compartidas por todas las instancias de la clase.

La cadena de prototipos, que fue el único mecanismo de herencia en versiones anteriores de ActionScript, sirve únicamente como función secundaria en ActionScript 3.0. El mecanismo de herencia principal, herencia de propiedades fijas, se administra internamente mediante el objeto traits. Una propiedad fija es una variable o un método que se define como parte de una definición de clase. La herencia de propiedades fijas también se denomina herencia de clase porque es el mecanismo de herencia asociado con palabras clave como `class`, `extends` y `override`.

La cadena de prototipos proporciona un mecanismo de herencia alternativo que es más dinámico que la herencia de propiedades fijas. Se pueden añadir propiedades a un objeto prototipo de la clase no sólo como parte de la definición de clase, sino también en tiempo de ejecución mediante la propiedad `prototype` del objeto de clase. Sin embargo, hay que tener en cuenta que si se establece el compilador en modo estricto, es posible que no se pueda acceder a propiedades añadidas a un objeto prototipo a menos que se declare una clase con la palabra clave `dynamic`.

La clase Object es un buen ejemplo de clase con varias propiedades asociadas al objeto prototipo. Los métodos `toString()` y `valueOf()` de la clase Object son en realidad funciones asignadas a propiedades del objeto prototipo de la clase Object. A continuación se muestra un ejemplo del aspecto que tendría en teoría la declaración de estos métodos (la implementación real difiere ligeramente a causa de los detalles de implementación):

```
public dynamic class Object
{
    prototype.toString = function()
    {
        // statements
    };
    prototype.valueOf = function()
    {
        // statements
    };
}
```

Como se mencionó anteriormente, se puede asociar una propiedad a un objeto prototipo de clase fuera de la definición de clase. Por ejemplo, el método `toString()` también se puede definir fuera de la definición de clase Object, de la manera siguiente:

```
Object.prototype.toString = function()
{
    // statements
};
```

Sin embargo, a diferencia de la herencia de propiedades fijas, la herencia de prototipo no requiere la palabra clave `override` si se desea volver a definir un método en una subclase. Por ejemplo, si se desea volver a definir el método `valueOf()` en una subclase de la clase Object, hay tres opciones. En primer lugar, se puede definir un método `valueOf()` en el objeto prototipo de la subclase, dentro de la definición de la clase. El código siguiente crea una subclase de Object denominada Foo y redefine el método `valueOf()` en el objeto prototipo de Foo como parte de la definición de clase. Como todas las clases heredan de Object, no es necesario utilizar la palabra clave `extends`.

```
dynamic class Foo
{
    prototype.valueOf = function()
    {
        return "Instance of Foo";
    };
}
```

En segundo lugar, se puede definir un método `valueOf()` en el objeto prototipo de Foo, fuera de la definición de clase, como se indica en el código siguiente:

```
Foo.prototype.valueOf = function()
{
    return "Instance of Foo";
};
```

Por último, se puede definir una propiedad fija denominada `valueOf()` como parte de la clase Foo. Esta técnica difiere de las otras en que mezcla la herencia de propiedades fijas con la herencia de prototipo. Cualquier subclase de Foo que vaya a redefinir `valueOf()` debe utilizar la palabra clave `override`. El código siguiente muestra `valueOf()` definido como una propiedad fija en Foo:

```
class Foo
{
    function valueOf():String
    {
        return "Instance of Foo";
    }
}
```

El espacio de nombres AS3

La existencia de dos mecanismos de herencia independientes, la herencia de propiedades fijas y la herencia de prototipo, crea un reto interesante para la compatibilidad con respecto a las propiedades y los métodos de las clases principales. La compatibilidad con la especificación del lenguaje ECMAScript en la que se basa ActionScript, requiere utilizar herencia de prototipo, lo que significa que las propiedades y los métodos de una clase principal se definen en el objeto prototipo de esa clase. Por otra parte, la compatibilidad con ActionScript 3.0 requiere utilizar la herencia de propiedades fijas, lo que significa que las propiedades y los métodos de una clase principal se definen en la definición de clase mediante las palabras clave `const`, `var` y `function`. Además, el uso de propiedades fijas en lugar de las versiones de prototipos puede proporcionar un aumento considerable de rendimiento en tiempo de ejecución.

ActionScript 3.0 resuelve este problema utilizando tanto la herencia de prototipo como la herencia de propiedades fijas para las clases principales. Cada clase principal contiene dos conjuntos de propiedades y métodos. Un conjunto se define en el objeto prototipo por compatibilidad con la especificación ECMAScript y el otro conjunto se define con propiedades fijas y el espacio de nombres AS3.0 por compatibilidad con ActionScript 3.0.

El espacio de nombres AS3 proporciona un cómodo mecanismo para elegir entre los dos conjuntos de propiedades y métodos. Si no se utiliza el espacio de nombres AS3, una instancia de una clase principal hereda las propiedades y métodos definidos en el objeto prototipo de la clase principal. Si se decide utilizar el espacio de nombres AS3, una instancia de una clase principal hereda las versiones de AS3, ya que siempre se prefieren las propiedades fijas a las propiedades de prototipo. Es decir, siempre que una propiedad fija está disponible, se utilizará en lugar de una propiedad de prototipo con el mismo nombre.

Se puede utilizar de forma selectiva la versión del espacio de nombres AS3 de una propiedad o un método calificándolo con el espacio de nombres AS3. Por ejemplo, el código siguiente utiliza la versión AS3 del método `Array.pop()`:

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // output: 1,2
```

Como alternativa, se puede utilizar la directiva `use namespace` para abrir el espacio de nombres AS3 para todas las definiciones de un bloque de código. Por ejemplo, el código siguiente utiliza la directiva `use namespace` para abrir el espacio de nombres AS3 para los métodos `pop()` y `push()`:

```
use namespace AS3;

var nums:Array = new Array(1, 2, 3);
nums.pop();
nums.push(5);
trace(nums) // output: 1,2,5
```

ActionScript 3.0 también proporciona opciones de compilador para cada conjunto de propiedades de forma que se pueda aplicar el espacio de nombres AS3 a todo el programa. La opción de compilador `-as3` representa el espacio de nombres AS3 y la opción de compilador `-es` representa la opción de herencia de prototipo (`es` significa ECMAScript). Para abrir el espacio de nombres AS3 para todo el programa, se debe establecer la opción de compilador `-as3` en `true` y la opción de compilador `-es` en `false`. Para utilizar las versiones de prototipos, las opciones de compilador deben establecerse en los valores opuestos. La configuración de compilador predeterminada para Adobe Flex Builder 3 y Adobe Flash CS4 Professional es `-as3 = true` y `-es = false`.

Si se pretende ampliar alguna de las clases principales y sustituir algún método, hay que comprender cómo puede afectar el espacio de nombres AS3 a la manera de declarar un método sustituido. Si se utiliza el espacio de nombres AS3, cualquier método sustituto de un método de clase principal también debe utilizar el espacio de nombres AS3, junto con el atributo `override`. Si no se utiliza el espacio de nombres AS3 y se desea redefinir un método de clase principal en una subclase, no se debe utilizar el espacio de nombres AS3 ni la palabra clave `override`.

Ejemplo: GeometricShapes

La aplicación de ejemplo GeometricShapes muestra cómo se pueden aplicar algunos conceptos y características de la orientación a objetos con ActionScript 3.0:

- Definición de clases
- Ampliación de clases
- Polimorfismo y la palabra clave `override`
- Definición, ampliación e implementación de interfaces

También incluye un "método de fábrica" que crea instancias de clase y muestra la manera de declarar un valor devuelto como una instancia de una interfaz y utilizar ese objeto devuelto de forma genérica.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación GeometricShapes se encuentran en la carpeta Samples/GeometricShapes. La aplicación consta de los siguientes archivos:

Archivo	**Descripción**
GeometricShapes.mxml o GeometricShapes.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/geometricshapes/IGeometricShape.as	Los métodos básicos de definición de interfaz que se van a implementar en todas las clases de la aplicación GeometricShapes.
com/example/programmingas3/geometricshapes/IPolygon.as	Una interfaz que define los métodos que se van a implementar en las clases de la aplicación GeometricShapes que tienen varios lados.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Un tipo de forma geométrica que tiene lados de igual longitud, simétricamente ubicados alrededor del centro de la forma.
com/example/programmingas3/geometricshapes/Circle.as	Un tipo de forma geométrica que define un círculo.

Archivo	Descripción
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Una subclase de RegularPolygon que define un triángulo con todos los lados de la misma longitud.
com/example/programmingas3/geometricshapes/Square.as	Una subclase de RegularPolygon que define un rectángulo con los cuatro lados de la misma longitud.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Una clase que contiene un método de fábrica para crear formas de un tipo y un tamaño específicos.

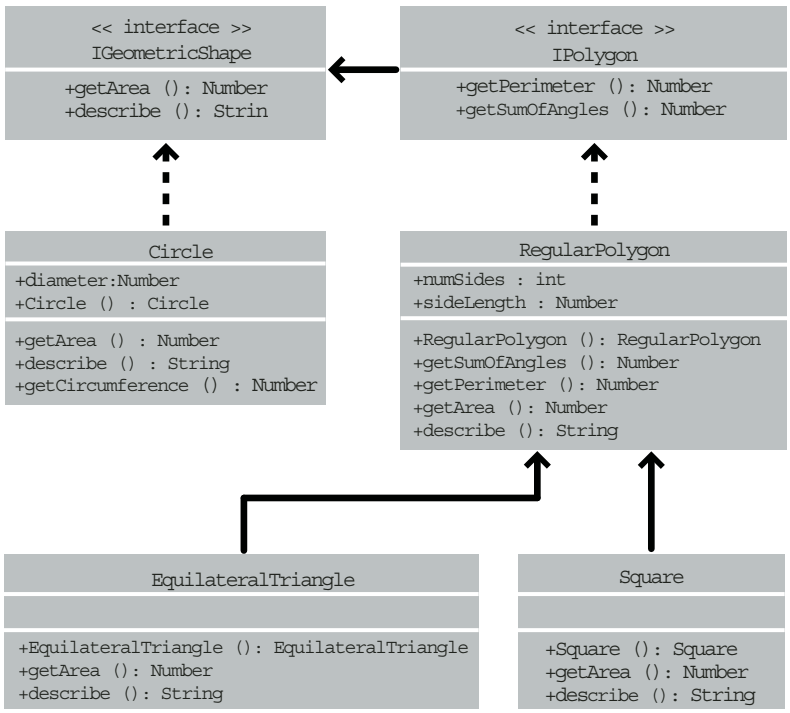
Definición de las clases de GeometricShapes

La aplicación GeometricShapes permite al usuario especificar un tipo de forma geométrica y un tamaño. Responde con una descripción de la forma, su área y su perímetro.

La interfaz de usuario de la aplicación es trivial: incluye algunos controles para seleccionar el tipo de forma, establecer el tamaño y mostrar la descripción. La parte más interesante de esta aplicación está bajo la superficie, en la estructura de las clases y las interfaces.

Esta aplicación manipula formas geométricas, pero no las muestra gráficamente. Proporciona un pequeña biblioteca de clases e interfaces que se volverán a utilizar en un ejemplo de un capítulo posterior (consulte "Ejemplo: SpriteArranger" en la página 323). El ejemplo SpriteArranger muestra las formas gráficamente y permite al usuario manipularlas, utilizando la arquitectura de clases proporcionada en la aplicación GeometricShapes.

Las clases e interfaces que definen las formas geométricas en este ejemplo se muestran en el diagrama siguiente con notación UML (Unified Modeling Language):



Clases de ejemplo GeometricShapes

Definición del comportamiento común en una interfaz

La aplicación GeometricShapes trata con tres tipos de formas: círculos, cuadrados y triángulos equiláteros. La estructura de la clase GeometricShapes empieza por una interfaz muy sencilla, IGeometricShape, que muestra métodos comunes a los tres tipos de formas:

```
package com.example.programmingas3.geometricshapes
{
    public interface IGeometricShape
    {
        function getArea():Number;
        function describe():String;
    }
}
```

La interfaz define dos métodos: `getArea()`, que calcula y devuelve el área de la forma y `describe()`, que crea una descripción textual de las propiedades de la forma.

También se pretende obtener el perímetro de cada forma. Sin embargo, el perímetro de un círculo es su circunferencia y se calcula de una forma única, por lo que en el caso del círculo el comportamiento es distinto del de un triángulo o un cuadrado. De todos modos, los triángulos, los cuadrados y otros polígonos son suficientemente similares, así que tiene sentido definir una nueva clase de interfaz para ellos: IPolygon. La interfaz IPolygon también es bastante sencilla, como se muestra a continuación:

```
package com.example.programmingas3.geometricshapes
{
    public interface IPolygon extends IGeometricShape
    {
        function getPerimeter():Number;
        function getSumOfAngles():Number;
    }
}
```

Esta interfaz define dos métodos comunes para todos los polígonos: el método `getPerimeter()` que mide la distancia combinada de todos los lados y el método `getSumOfAngles()` que suma todos los ángulos interiores.

La interfaz IPolygon amplía la interfaz IGeometricShape, lo que significa que cualquier clase que implemente la interfaz IPolygon debe declarar los cuatro métodos: dos de la interfaz IGeometricShape y dos de la interfaz IPolygon.

Definición de las clases de formas

Cuando ya se conozcan los métodos comunes a cada tipo de forma, se pueden definir las clases de las formas. Por la cantidad de métodos que hay que implementar, la forma más sencilla es la clase Circle, que se muestra a continuación:

```
package com.example.programmingas3.geometricshapes
{
    public class Circle implements IGeometricShape
    {
        public var diameter:Number;

        public function Circle(diam:Number = 100):void
        {
            this.diameter = diam;
        }

        public function getArea():Number
        {
            // The formula is Pi * radius * radius.
            var radius:Number = diameter / 2;
            return Math.PI * radius * radius;
        }

        public function getCircumference():Number
        {
            // The formula is Pi * diameter.
            return Math.PI * diameter;
        }

        public function describe():String
        {
            var desc:String = "This shape is a Circle.\n";
            desc += "Its diameter is " + diameter + " pixels.\n";
            desc += "Its area is " + getArea() + ".\n";
            desc += "Its circumference is " + getCircumference() + ".\n";
            return desc;
        }
    }
}
```

La clase Circle implementa la interfaz IGeometricShape, por lo que hay que proporcionar código para el método `getArea()` y el método `describe()`. Además, define el método `getCircumference()`, que es único para la clase Circle. La clase Circle también declara una propiedad, `diameter`, que no se encuentra en las clases de los otros polígonos.

Los otros dos tipos de formas, cuadrados y triángulos equiláteros, tienen algunos aspectos en común: cada uno de ellos tiene lados de longitud simular y existen fórmulas comunes que se pueden utilizar para calcular el perímetro y la suma de los ángulos interiores para ambos. De hecho, esas fórmulas comunes se aplicarán a cualquier otro polígono regular que haya que definir en el futuro.

La clase RegularPolygon será la superclase para la clase Square y la clase EquilateralTriangle. Una superclase permite centralizar la definición de métodos comunes de forma que no sea necesario definirlos por separado en cada subclase. A continuación se muestra el código para la clase RegularPolygon:

```
package com.example.programmingas3.geometricshapes
{
    public class RegularPolygon implements IPolygon
    {
        public var numSides:int;
        public var sideLength:Number;

        public function RegularPolygon(len:Number = 100, sides:int = 3):void
        {
            this.sideLength = len;
            this.numSides = sides;
        }

        public function getArea():Number
        {
            // This method should be overridden in subclasses.
            return 0;
        }

        public function getPerimeter():Number
        {
            return sideLength * numSides;
        }

        public function getSumOfAngles():Number
        {
            if (numSides >= 3)
            {
                return ((numSides - 2) * 180);
            }
            else
            {
                return 0;
            }
        }

        public function describe():String
        {
            var desc:String = "Each side is " + sideLength + " pixels long.\n";
            desc += "Its area is " + getArea() + " pixels square.\n";
            desc += "Its perimeter is " + getPerimeter() + " pixels long.\n";
            desc += "The sum of all interior angles in this shape is " + getSumOfAngles() + "
degrees.\n";
            return desc;
        }
    }
}
```

En primer lugar, la clase RegularPolygon declara dos propiedades que son comunes a todos los polígonos regulares: la longitud de cada lado (propiedad `sideLength`) y el número de lados (propiedad `numSides`).

La clase RegularPolygon implementa la interfaz IPolygon y declara los cuatro métodos de la interfaz IPolygon. Implementa dos de ellos, `getPerimeter()` y `getSumOfAngles()`, utilizando fórmulas comunes.

Como la fórmula para el método `getArea()` varía de una forma a otra, la versión de la clase base del método no puede incluir lógica común que pueda ser heredada por los métodos de la subclase. Sólo devuelve el valor predeterminado 0, para indicar que no se ha calculado el área. Para calcular el área de cada forma correctamente, las subclases de la clase RegularPolygon tendrán que sustituir el método `getArea()`.

El código siguiente para la clase EquilateralTriangle muestra cómo se sustituye el método `getArea()`:

```
package com.example.programmingas3.geometricshapes
{
    public class EquilateralTriangle extends RegularPolygon
    {
        public function EquilateralTriangle(len:Number = 100):void
        {
            super(len, 3);
        }

        public override function getArea():Number
        {
        // The formula is ((sideLength squared) * (square root of 3)) / 4.
        return ( (this.sideLength * this.sideLength) * Math.sqrt(3) ) / 4;
        }

        public override function describe():String
        {
                /* starts with the name of the shape, then delegates the rest
                of the description work to the RegularPolygon superclass */
        var desc:String = "This shape is an equilateral Triangle.\n";
        desc += super.describe();
        return desc;
        }
    }
}
```

La palabra clave `override` indica que el método `EquilateralTriangle.getArea()` sustituye de forma intencionada el método `getArea()` de la superclase RegularPolygon. Cuando se llama al método `EquilateralTriangle.getArea()`, se calcula el área con la fórmula del fragmento de código anterior y no se ejecuta el código del método `RegularPolygon.getArea()`.

En cambio, la clase EquilateralTriangle no define su propia versión del método `getPerimeter()`. Cuando se llama al método `EquilateralTriangle.getPerimeter()`, la llamada sube por la cadena de herencia y ejecuta el código del método `getPerimeter()` de la superclase RegularPolygon.

El constructor de `EquilateralTriangle()` utiliza la sentencia `super()` para invocar explícitamente el constructor de `RegularPolygon()` de su superclase. Si ambos constructores tuvieran el mismo conjunto de parámetros, se podría omitir el constructor de `EquilateralTriangle()` y se ejecutaría en su lugar el constructor de `RegularPolygon()`. No obstante, el constructor de `RegularPolygon()` requiere un parámetro adicional, `numSides`. Así, el constructor de `EquilateralTriangle()` llama a `super(len, 3)`, que pasa el parámetro de entrada `len` y el valor 3 para indicar que el triángulo tendrá 3 lados.

El método `describe()` también utiliza la sentencia `super()` (aunque de una manera diferente) para invocar la versión del método `describe()` de la superclase RegularPolygon. El método `EquilateralTriangle.describe()` establece primero la variable de cadena `desc` en una declaración del tipo de forma. A continuación, obtiene los resultados del método `RegularPolygon.describe()` llamando a `super.describe()` y añade el resultado a la cadena `desc`.

No se proporciona en esta sección una descripción detallada de la clase Square, pero es similar a la clase EquilateralTriangle; proporciona un constructor y su propia implementación de los métodos `getArea()` y `describe()`.

Polimorfismo y el método de fábrica

Un conjunto de clases que haga buen uso de las interfaces y la herencia se puede utilizar de muchas maneras interesantes. Por ejemplo, todas las clases de formas descritas hasta ahora implementan la interfaz IGeometricShape o amplían una superclase que lo hace. Así, si se define una variable como una instancia de IGeometricShape, no hay que saber si es realmente una instancia de la clase Circle o de la clase Square para llamar a su método `describe()`.

El código siguiente muestra cómo funciona esto:

```
var myShape:IGeometricShape = new Circle(100);
trace(myShape.describe());
```

Cuando se llama a `myShape.describe()`, se ejecuta el método `Circle.describe()` ya que, aunque la variable se define como una instancia de la interfaz IGeometricShape, Circle es su clase subyacente.

En este ejemplo se muestra el principio del polimorfismo en activo: la misma llamada al método tiene como resultado la ejecución de código diferente, dependiendo de la clase del objeto cuyo método se esté invocando.

La aplicación GeometricShapes aplica este tipo de polimorfismo basado en interfaz con una versión simplificada de un patrón de diseño denominado método de fábrica. El término *método de fábrica* hace referencia a una función que devuelve un objeto cuyo tipo de datos o contenido subyacente puede variar en función del contexto.

La clase GeometricShapeFactory mostrada define un método de fábrica denominado `createShape()`:

```
package com.example.programmingas3.geometricshapes
{
    public class GeometricShapeFactory
    {
        public static var currentShape:IGeometricShape;

        public static function createShape(shapeName:String,

len:Number):IGeometricShape
        {
            switch (shapeName)
            {
                case "Triangle":
                    return new EquilateralTriangle(len);

                case "Square":
                    return new Square(len);

                case "Circle":
                    return new Circle(len);
            }
            return null;
        }

        public static function describeShape(shapeType:String, shapeSize:Number):String
        {
            GeometricShapeFactory.currentShape =
                GeometricShapeFactory.createShape(shapeType, shapeSize);
            return GeometricShapeFactory.currentShape.describe();
        }
    }
}
```

El método de fábrica `createShape()` permite a los constructores de subclases de formas definir los detalles de las instancias que crean, devolviendo los objetos nuevos como instancias de IGeometricShape de forma que puedan ser manipulados por la aplicación de una manera más general.

El método `describeShape()` del ejemplo anterior muestra cómo se puede utilizar el método de fábrica en una aplicación para obtener una referencia genérica a un objeto más específico. La aplicación puede obtener la descripción para un objeto Circle recién creado así:

```
GeometricShapeFactory.describeShape("Circle", 100);
```

A continuación, el método `describeShape()` llama al método de fábrica `createShape()` con los mismos parámetros y almacena el nuevo objeto Circle en una variable estática denominada `currentShape`, a la que se le asignó el tipo de un objeto IGeometricShape. A continuación, se llama al método `describe()` en el objeto `currentShape` y se resuelve esa llamada automáticamente para ejecutar el método `Circle.describe()`, lo que devuelve una descripción detallada del círculo.

Mejora de la aplicación de ejemplo

La verdadera eficacia de las interfaces y la herencia se aprecia al ampliar o cambiar la aplicación.

Por ejemplo, se puede añadir una nueva forma (un pentágono) a esta aplicación de ejemplo. Para ello se crea una nueva clase Pentagon que amplía la clase RegularPolygon y define sus propias versiones de los métodos `getArea()` y `describe()`. A continuación, se añade una nueva opción Pentagon al cuadro combinado de la interfaz de usuario de la aplicación. Y ya está. La clase Pentagon recibirá automáticamente la funcionalidad de los métodos `getPerimeter()` y `getSumOfAngles()` de la clase RegularPolygon por herencia. Como hereda de una clase que implementa la interfaz IGeometricShape, una instancia de Pentagon puede tratarse también como una instancia de IGeometricShape. Esto significa que para agregar un nuevo tipo de forma, no es necesario cambiar la firma del método de ningún método en la clase GeometricShapeFactory (y por lo tanto, tampoco no es necesario modificar ninguna parte de código que utilice la clase GeometricShapeFactory).

Se puede añadir una clase Pentagon al ejemplo Geometric Shapes como ejercicio, para ver cómo facilitan las interfaces y la herencia el trabajo de añadir nuevas características a una aplicación.

Capítulo 6: Trabajo con fechas y horas

Puede ser que el tiempo no sea lo más importante, pero suele ser un factor clave en las aplicaciones de software. ActionScript 3.0 proporciona formas eficaces de administrar fechas de calendario, horas e intervalos de tiempo. Dos clases principales proporcionan la mayor parte de esta funcionalidad de tiempo: la cases Date y Timer del paquete flash.utils.

Fundamentos de la utilización de fechas y horas

Introducción a la utilización de fechas y horas

Las fechas y las horas son un tipo de información utilizado frecuentemente en programas escritos en ActionScript. Por ejemplo, puede ser necesario averiguar el día de la semana o medir cuánto tiempo pasa un usuario en una pantalla determinada, entre otras muchas cosas. En ActionScript se puede utilizar la clase Date para representar un momento puntual en el tiempo, incluida la información de fecha y hora. En una instancia de Date hay valores para las unidades individuales de fecha y hora, incluidas año, mes, fecha, día de la semana, hora, minutos, segundos, milisegundos y zona horaria. Para usos más avanzados, ActionScript también incluye la clase Timer, que se puede utilizar para realizar acciones después de un retardo o a intervalos repetidos.

Tareas comunes relacionadas con fechas y horas

En este capítulo se describen las siguientes tareas comunes para trabajar con información de fecha y hora:

- Trabajo con objetos Date
- Obtención de la fecha y la hora actuales
- Acceso a unidades de fecha y hora individuales (días, años, horas, minutos, etc.)
- Realización de cálculos aritméticos con fechas y horas
- Conversión entre zonas horarias
- Realización de acciones periódicas
- Realización de acciones después de un intervalo de tiempo establecido

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Hora UTC: hora universal coordinada, la zona horaria de referencia. Todas las demás zonas horarias se definen como un número de horas (más o menos) de diferencia con respecto a la hora UTC.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Como los listados de código de este capítulo se centran principalmente en los objetos Date, la ejecución de los ejemplos implica ver los valores de las variables utilizadas en los ejemplos, ya sea escribiendo valores en una instancia de campo de texto del objeto Stage o utilizando la función `trace()` para imprimir valores en el panel Salida. Estas técnicas se describen detalladamente en “Prueba de los listados de código de ejemplo del capítulo” en la página 36.

Administración de fechas de calendario y horas

Todas las funciones de administración de fechas de calendario y horas de ActionScript 3.0 se concentran en la clase Date de nivel superior. La clase Date contiene métodos y propiedades que permiten gestionar fechas y horas en la hora universal (UTC) o en la hora local específica de una zona horaria. La hora universal (UTC) es una definición de hora estándar que básicamente coincide con la hora del meridiano de Greenwich (GMT).

Creación de objetos Date

El método constructor de la clase Date es uno de los métodos constructores de clase principal más versátiles. Se puede llamar de cuatro formas distintas.

En primer lugar, si no se proporcionan parámetros, el constructor `Date()` devuelve un objeto Date que contiene la fecha y hora actuales, en la hora local correspondiente a la zona horaria. A continuación se muestra un ejemplo:

```
var now:Date = new Date();
```

En segundo lugar, si se proporciona un solo parámetro numérico, el constructor `Date()` trata dicho parámetro como el número de milisegundos transcurridos desde el 1 de enero de 1970 y devuelve un objeto Date correspondiente. Hay que tener en cuenta que el valor de milisegundos que se pasa se trata como milisegundos desde el 1 de enero de 1970, en UTC. Sin embargo, el objeto Date muestra los valores en la zona horaria local, a menos que se utilicen métodos específicos de UTC para recuperarlos y mostrarlos. Si se crea un nuevo objeto Date con un solo parámetro de milisegundos, hay tener en cuenta la diferencia de zona horaria entre la hora local y la hora universal UTC. Las siguientes sentencias crean un objeto Date establecido en la medianoche del día 1 de enero de 1970, en UTC:

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// gets a Date one day after the start date of 1/1/1970
var startTime:Date = new Date(millisecondsPerDay);
```

En tercer lugar, se pueden pasar varios parámetros numéricos al constructor `Date()`. Estos parámetros se tratan como el año, mes, día, hora, minuto, segundo y milisegundo, respectivamente, y se devuelve un objeto Date correspondiente. Se supone que estos parámetros de entrada están en hora local y no en UTC. Las siguientes sentencias obtienen un objeto Date establecido en la medianoche del inicio del día 1 de enero de 2000, en hora local:

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

En cuarto lugar, se puede pasar un solo parámetro de cadena al constructor `Date()`. Éste intentará analizar dicha cadena en componentes de fecha u hora, y devolver a continuación un objeto Date correspondiente. Si utiliza este enfoque, es una buena idea incluir el constructor `Date()` en un bloque `try..catch` para localizar cualquier error de análisis. El constructor `Date()` acepta varios formatos de cadena, tal y como se especifica en la Referencia del lenguaje y componentes ActionScript 3.0. La siguiente sentencia inicializa un nuevo objeto Date con un valor de cadena:

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Si el constructor `Date()` no puede analizar correctamente el parámetro de cadena, no emitirá una excepción. Sin embargo, el objeto Date resultante contendrá un valor de fecha no válido.

Obtención de valores de unidad de tiempo

Es posible extraer los valores de diversas unidades de tiempo de un objeto Date a través de las propiedades y métodos de la clase Date. Cada una de las siguientes propiedades asigna el valor de una unidad de tiempo en el objeto Date:

- La propiedad `fullYear`
- La propiedad `month`, que tiene un formato numérico con un valor entre 0 (enero) y 11 (diciembre)
- La propiedad `date`, que es el número de calendario del día del mes, en el rango de 1 a 31

- La propiedad `day`, que es el día de la semana en formato numérico, siendo 0 el valor correspondiente al domingo
- La propiedad `hours`, en el rango de 0 a 23
- La propiedad `minutes`
- La propiedad `seconds`
- La propiedad `milliseconds`

  De hecho, la clase Date proporciona varias formas de obtener cada uno de estos valores. Por ejemplo, se puede obtener el valor de mes de un objeto Date de cuatro formas distintas:

- La propiedad `month`
- El método `getMonth()`
- La propiedad `monthUTC`
- El método `getMonthUTC()`

  Los cuatro métodos son igualmente eficaces, de forma que puede elegirse el que mejor convenga a cada aplicación.

  Todas las propiedades anteriormente enumeradas representan componentes del valor de fecha total. Por ejemplo, el valor de la propiedad milliseconds nunca será mayor que 999, ya que cuando llega a 1000, el valor de los segundos aumenta en 1 y el valor de la propiedad milliseconds se restablece en 0.

  Si se desea obtener el valor del objeto Date en milisegundos transcurridos desde el 1 de enero de 1970 (UTC), se puede utilizar el método `getTime()`. El método opuesto, `setTime()`, permite cambiar el valor de un objeto Date existente utilizando los milisegundos transcurridos desde el 1 de enero de 1970 (UTC).

Operaciones aritméticas de fecha y hora

La clase Date permite realizar sumas y restas en fechas y horas. Los valores de fecha se almacenan internamente como milisegundos, de forma que es necesario convertir los demás valores a milisegundos para poder realizar sumas o restas en objetos Date.

Si la aplicación va a realizar muchas operaciones aritméticas de fecha y hora, quizás resulte útil crear constantes que conserven valores comunes de unidad de tiempo en milisegundos, como se muestra a continuación:

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

Ahora resulta sencillo realizar operaciones aritméticas de fecha con unidades de tiempo estándar. El código siguiente establece un valor de fecha de una hora a partir de la hora actual, mediante los métodos `getTime()` y `setTime()`:

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

Otra forma de establecer un valor de fecha consiste en crear un nuevo objeto Date con un solo parámetro de milisegundos. Por ejemplo, el siguiente código añade 30 días a una fecha para calcular otra:

```
// sets the invoice date to today's date
var invoiceDate:Date = new Date();

// adds 30 days to get the due date
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

A continuación, se multiplica por 30 la constante `millisecondsPerDay` para representar un tiempo de 30 días y el resultado se añade al valor `invoiceDate` y se utiliza para establecer el valor `dueDate`.

Conversión entre zonas horarias

Las operaciones aritméticas de fecha y hora son útiles para convertir fechas de una zona horaria a otra. Esta conversión se realiza a través del método `getTimezoneOffset()`, que devuelve el valor en minutos de diferencia entre la zona horaria del objeto Date y la hora universal UTC. Devuelve un valor en minutos porque no todas las zonas horarias se establecen mediante incrementos de horas completas, sino que en algunos casos existen diferencias de media hora entre zonas vecinas.

En el siguiente ejemplo se utiliza el desplazamiento de zona horaria para convertir una fecha de la hora local a la hora universal UTC. Para realizar la conversión, primero se calcula el valor de zona horaria en milisegundos y luego se ajusta el valor de Date en dicha cantidad:

```
// creates a Date in local time
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// converts the Date to UTC by adding or subtracting the time zone offset
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

Control de intervalos de tiempo

Cuando se desarrollan aplicaciones con Adobe Flash CS4 Professional, se tiene acceso a la línea de tiempo, que proporciona una progresión uniforme, fotograma a fotograma, a través de la aplicación. Sin embargo, en proyectos sólo de ActionScript, hay que confiar en otros mecanismos de tiempo.

Bucles frente a temporizadores

En algunos lenguajes de programación, es necesario crear esquemas temporales propios mediante sentencias de bucle como`for` o `do..while`.

Las sentencias de bucle suelen ejecutarse con la máxima rapidez permitida en el equipo, lo que significa que la aplicación se ejecuta más rápidamente en unos equipos que en otros. Si una aplicación debe tener un intervalo de tiempo coherente, es necesario asociarla a un calendario o reloj real. Muchas aplicaciones como juegos, animaciones y controladores de tiempo real necesitan mecanismos de tictac regulares que sean coherentes de un equipo a otro.

La clase Timer de ActionScript 3.0 proporciona una solución eficaz. A través del modelo de eventos de ActionScript 3.0, la clase Timer distribuye eventos del temporizador cuando se alcanza un intervalo de tiempo especificado.

La clase Timer

La forma preferida de gestionar funciones de tiempo en ActionScript 3.0 consiste en utilizar la clase Timer (flash.utils.Timer) para distribuir eventos cuando se alcanza un intervalo.

Para iniciar un temporizador, primero es necesario crear una instancia de la clase Timer e indicarle la frecuencia con la que debe generar un evento del temporizador y la cantidad de veces que debe hacerlo antes de detenerse.

Por ejemplo, el código siguiente crea una instancia de Timer que distribuye un evento cada segundo y se prolonga durante 60 segundos:

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

El objeto Timer distribuye un objeto TimerEvent cada vez que se alcanza el intervalo especificado. El tipo de evento de un objeto TimerEvent es `timer` (definido por la constante `TimerEvent.TIMER`). Un objeto TimerEvent contiene las mismas propiedades que un objeto Event estándar.

Si la instancia de Timer se establece en un número fijo de intervalos, también distribuirá un evento `timerComplete` (definido por la constante `TimerEvent.TIMER_COMPLETE`) cuando alcance el intervalo final.

A continuación se muestra una aplicación de ejemplo donde se utiliza la clase Timer:

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // creates a new five-second Timer
            var minuteTimer:Timer = new Timer(1000, 5);

            // designates listeners for the interval and completion events
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE, onTimerComplete);

            // starts the timer ticking
            minuteTimer.start();
        }

        public function onTick(event:TimerEvent):void
        {
            // displays the tick count so far
            // The target of this event is the Timer instance itself.
            trace("tick " + event.target.currentCount);
        }

        public function onTimerComplete(event:TimerEvent):void
        {
            trace("Time's Up!");
        }
    }
}
```

Al crear la clase ShortTimer, se crea una instancia de Timer que hará tictac una vez por segundo durante cinco segundos. Posteriormente agrega dos detectores al temporizador: uno que detecta todos los tictac y otro que detecta el evento `timerComplete` .

A continuación, inicia el tictac del temporizador y, a partir de ese punto, el método `onTick()` se ejecuta en intervalos de un segundo.

El método `onTick()` simplemente muestra el recuento de tictac actual. Después de cinco segundos, se ejecuta el método `onTimerComplete()`, que indica que se ha acabado el tiempo.

Cuando se ejecuta este ejemplo, aparecen las siguientes líneas en la consola o ventana de traza a una velocidad de una línea por segundo:

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

Funciones de tiempo en el paquete flash.utils

ActionScript 3.0 incluye una serie de funciones de tiempo similares a las que estaban disponibles en ActionScript 2.0. Estas funciones se proporcionan como funciones de nivel de paquete en flash.utils y funcionan del mismo modo que en ActionScript 2.0.

Función	Descripción
`clearInterval(id:uint):void`	Cancela una llamada a `setInterval()` especificada.
`clearTimeout(id:uint):void`	Cancela una llamada `setTimeout()` especificada.
`getTimer():int`	Devuelve el número de milisegundos transcurridos desde que se inició Adobe® Flash® Player o Adobe® AIR™.
`setInterval(closure:Function, delay:Number, ... arguments):uint`	Ejecuta una función en un intervalo especificado (en milisegundos).
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	Ejecuta una función especificada tras una demora especificada (en milisegundos).

Estas funciones se conservan en ActionScript 3.0 para permitir la compatibilidad con versiones anteriores. Adobe no recomienda utilizarlas en nuevas aplicaciones de ActionScript 3.0. En general, resulta más sencillo y eficaz utilizar la clase Timer en las aplicaciones.

Ejemplo: Sencillo reloj analógico

En este ejemplo de un sencillo reloj analógico se ilustran dos de los conceptos de fecha y hora tratados en este capítulo:

- Obtención de la fecha y hora actuales, y extracción de valores para las horas, minutos y segundos
- Utilización de un objeto Timer para establecer el ritmo de una aplicación

  Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación SimpleClock se encuentran en la carpeta Samples/SimpleClock. La aplicación consta de los siguientes archivos:

Archivo	Descripción
SimpleClockApp.mxml o SimpleClockApp.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/simpleclock/SimpleClock.as	El archivo de la aplicación principal.
com/example/programmingas3/simpleclock/AnalogClockFace.as	Dibuja una esfera de reloj redonda y las manecillas de hora, minutos y segundos, en función de la hora.

Definición de la clase SimpleClock

El ejemplo del reloj es sencillo, pero es recomendable organizar bien incluso las aplicaciones sencillas para poder ampliarlas fácilmente en el futuro. En ese sentido, la aplicación SimpleClock utiliza la clase SimpleClock para gestionar las tareas de inicio y mantenimiento de hora, y luego utiliza otra clase denominada AnalogClockFace para mostrar realmente la hora.

A continuación se muestra el código que define e inicializa la clase SimpleClock (hay que tener en cuenta que, en la versión de Flash, SimpleClock amplía la clase Sprite):

```
public class SimpleClock extends UIComponent
{
    /**
     * The time display component.
     */
    private var face:AnalogClockFace;

    /**
     * The Timer that acts like a heartbeat for the application.
     */
    private var ticker:Timer;
```

La clase tiene dos propiedades importantes:

- La propiedad `face`, que es una instancia de la clase AnalogClockFace
- La propiedad `ticker`, que es una instancia de la clase Timer

  La clase SimpleClock utiliza un constructor predeterminado. El método `initClock()` realiza el trabajo de configuración real, pues crea la esfera del reloj e inicia el tictac de la instancia de Timer.

Creación de la esfera del reloj

Las siguientes líneas del código de SimpleClock crean la esfera del reloj que se utiliza para mostrar la hora:

```
    /**
     * Sets up a SimpleClock instance.
     */
    public function initClock(faceSize:Number = 200)
    {
        // creates the clock face and adds it to the display list
        face = new AnalogClockFace(Math.max(20, faceSize));
        face.init();
        addChild(face);

        // draws the initial clock display
        face.draw();
```

El tamaño de la esfera puede pasarse al método `initClock()`. Si no se pasa el valor `faceSize`, se utiliza un tamaño predeterminado de 200 píxeles.

A continuación, la aplicación inicializa la esfera y la añade a la lista de visualización a través del método `addChild()` heredado de la clase DisplayObject. Luego llama al método `AnalogClockFace.draw()` para mostrar la esfera del reloj una vez, con la hora actual.

Inicio del temporizador

Después de crear la esfera del reloj, el método `initClock()` configura un temporizador:

```
    // creates a Timer that fires an event once per second
    ticker = new Timer(1000);

    // designates the onTick() method to handle Timer events
    ticker.addEventListener(TimerEvent.TIMER, onTick);

    // starts the clock ticking
    ticker.start();
```

En primer lugar, este método crea una instancia de Timer que distribuirá un evento una vez por segundo (cada 1000 milisegundos). Como no se pasa ningún parámetro `repeatCount` al constructor `Timer()`, el objeto Timer se repetirá indefinidamente.

El método `SimpleClock.onTick()` se ejecutará una vez por segundo cuando se reciba el evento `timer`:

```
public function onTick(event:TimerEvent):void
{
    // updates the clock display
    face.draw();
}
```

El método `AnalogClockFace.draw()` simplemente dibuja la esfera del reloj y las manecillas.

Visualización de la hora actual

La mayoría del código de la clase AnalogClockFace se utiliza para configurar los elementos que se visualizan en la esfera del reloj. Cuando se inicializa AnalogClockFace, dibuja un contorno circular, coloca una etiqueta de texto numérico en cada marca de hora y luego crea tres objetos Shape: uno para la manecilla de las horas, otro para la de los minutos y un tercero para la de los segundos.

Cuando se ejecuta la aplicación SimpleClock, llama al método `AnalogClockFace.draw()` cada segundo, del siguiente modo:

```
/**
 * Called by the parent container when the display is being drawn.
 */
public override function draw():void
{
    // stores the current date and time in an instance variable
    currentTime = new Date();
    showTime(currentTime);
}
```

Este método guarda la hora actual en una variable, de forma que la hora no puede cambiar mientras se dibujan las manecillas del reloj. Luego llama al método `showTime()` para mostrar las manecillas, como se muestra a continuación:

```
/**
 * Displays the given Date/Time in that good old analog clock style.
 */
public function showTime(time:Date):void
{
    // gets the time values
    var seconds:uint = time.getSeconds();
    var minutes:uint = time.getMinutes();
    var hours:uint = time.getHours();

    // multiplies by 6 to get degrees
    this.secondHand.rotation = 180 + (seconds * 6);
    this.minuteHand.rotation = 180 + (minutes * 6);

    // Multiply by 30 to get basic degrees, then
    // add up to 29.5 degrees (59 * 0.5)
    // to account for the minutes.
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

En primer lugar, este método extrae los valores de las horas, minutos y segundos de la hora actual. Luego utiliza estos valores para calcular el ángulo de cada manecilla. Como la manecilla de los segundos realiza una rotación completa en 60 segundos, gira 6 grados cada segundo (360/60). La manecilla de los minutos gira en la misma medida cada minuto.

La manecilla de las horas se actualiza también cada minuto, de forma que va progresando a medida que pasan los minutos. Gira 30 grados cada hora (360/12), pero también gira medio grado cada minuto (30 grados divididos entre 60 minutos).

Capítulo 7: Trabajo con cadenas

La clase String contiene métodos que le permiten trabajar con cadenas de texto. Las cadenas son importantes para trabajar con muchos de los objetos. Los métodos descritos en este capítulo son útiles para trabajar con cadenas utilizadas en objetos como TextField, StaticText, XML, ContextMenu y FileReference.

Las cadenas son secuencias de caracteres. ActionScript 3.0 admite caracteres ASCII y Unicode.

Fundamentos de la utilización de cadenas

Introducción a la utilización de cadenas

En jerga de programación, una cadena es un valor de texto: una secuencia de letras, números u otros caracteres agrupados en un solo valor. Por ejemplo, esta línea de código crea una variable con el tipo de datos String y asigna un valor de literal de cadena a esa variable:

```
var albumName:String = "Three for the money";
```

Como se ilustra en este ejemplo, en ActionScript se puede denotar un valor de cadena escribiendo texto entre comillas dobles o simples. A continuación se muestran varios ejemplos de cadenas:

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

Siempre que se manipule un fragmento de texto en ActionScript, se trabaja con un valor de cadena. La clase String de ActionScript es el tipo de datos que se utiliza para trabajar con valores de texto. Se suelen utilizar instancias de String para propiedades, parámetros de métodos, etc. en muchas otras clases de ActionScript.

Tareas comunes para la utilización de cadenas

A continuación se enumeran algunas tareas comunes relacionadas con cadenas que se describen en este capítulo:

- Creación de objetos String
- Trabajo con caracteres especiales como el retorno de carro, la tabulación y caracteres no representados en el teclado
- Medición de la longitud de cadenas
- Aislamiento de caracteres individuales de una cadena
- Unión de cadenas
- Comparación de cadenas
- Búsqueda, extracción y reemplazo de partes de una cadena
- Conversión de cadenas a mayúsculas o minúsculas

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- ASCII: sistema para representar caracteres de texto y símbolos en programas informáticos. El sistema ASCII incluye el alfabeto inglés de 26 letras y un conjunto limitado de caracteres adicionales.

- Carácter: unidad más pequeña de los datos de texto (una letra o un símbolo).
- Concatenación: unión de varios valores de cadena añadiendo uno a continuación del otro para crear un nuevo valor de cadena.
- Cadena vacía: cadena que no contiene texto, espacio en blanco ni otros caracteres; se escribe como `""`. Un valor de cadena vacía no es lo mismo que una variable de tipo String con valor NULL. Una variable de tipo String con valor NULL es una variable que no tiene una instancia de String asignada, mientras que una cadena vacía tiene una instancia con un valor que no contiene ningún carácter.
- Cadena: valor de texto (secuencia de caracteres).
- Literal de cadena (o "cadena literal"): valor de cadena escrito explícitamente en el código como un valor de texto entre comillas dobles o simples.
- Subcadena: cadena que forma parte de otra cadena.
- Unicode: sistema estándar para representar caracteres de texto y símbolos en programas informáticos. El sistema Unicode permite utilizar cualquier carácter de un sistema de escritura.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Como los listados de código de este capítulo se centran principalmente en la manipulación de texto, la ejecución de los ejemplos implica ver los valores de las variables utilizadas en los ejemplos, ya sea escribiendo valores en una instancia de campo de texto del objeto Stage o utilizando la función `trace()` para imprimir valores en el panel Salida. Estas técnicas se describen de forma detallada en "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

Creación de cadenas

La clase String se utiliza para representar datos de cadena (textuales) en ActionScript 3.0. Las cadenas de ActionScript admiten caracteres tanto ASCII como Unicode. La manera más sencilla de crear una cadena es utilizar un literal de cadena. Para declarar un literal de cadena, hay que utilizar comilla dobles rectas (`"`) o comillas simples (`'`). Por ejemplo, las dos cadenas siguientes son equivalentes:

```
var str1:String = "hello";
var str2:String = 'hello';
```

También se puede declarar una cadena utilizando el operador `new`, de la manera siguiente:

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

Las dos cadenas siguientes son equivalentes:

```
var str1:String = "hello";
var str2:String = new String("hello");
```

Para utilizar comillas simples (`'`) dentro de un literal de cadena definido con comillas simples (`'`) hay que utilizar el carácter de escape barra diagonal inversa (`\`). De manera similar, para utilizar comillas dobles (`"`) dentro de un literal de cadena definido con comillas dobles (`"`) hay que utilizar el carácter de escape barra diagonal inversa (`\`). Las dos cadenas siguientes son equivalentes:

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

Se puede elegir utilizar comillas simples o comillas dobles en función de las comillas simples o dobles que existan en un literal de cadena, como se muestra a continuación:

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

Hay que tener en cuenta que ActionScript distingue entre una comilla simple recta (') y una comilla simple izquierda o derecha (' o '). Lo mismo ocurre para las comillas dobles. Hay que utilizar comillas rectas para delimitar los literales de cadena. Al pegar texto de otro origen en ActionScript hay que asegurarse de utilizar los caracteres correctos.

Como se muestra en la tabla siguiente, se puede utilizar el carácter de escape de barra diagonal inversa (\) para definir otros caracteres en los literales de cadena:

Secuencia de escape	Carácter
\b	Retroceso
\f	Salto de página
\n	Nueva línea
\r	Retorno de carro
\t	Tabulador
\u***nnnn***	Carácter Unicode con el código de carácter especificado por el número hexadecimal *nnnn*; por ejemplo, \u263a es el carácter smiley.
\\x***nn***	Carácter ASCII con el código de carácter especificado por el número hexadecimal *nn*.
\'	Comilla simple
\"	Comilla doble
\\	Barra diagonal inversa simple

La propiedad length

Cada cadena tiene una propiedad `length` que es igual al número de caracteres de la cadena:

```
var str:String = "Adobe";
trace(str.length);            // output: 5
```

Tanto la cadena vacía como la cadena NULL tienen longitud 0, como se muestra en el siguiente ejemplo:

```
var str1:String = new String();
trace(str1.length);           // output: 0

str2:String = '';
trace(str2.length);           // output: 0
```

Trabajo con caracteres en cadenas

Cada carácter de una cadena tiene una posición de índice en la cadena (un entero). La posición de índice del primer carácter es 0. Por ejemplo, en la siguiente cadena, el carácter `y` está en la posición 0 y el carácter `w` en la posición 5:

```
"yellow"
```

Se pueden examinar caracteres individuales en diversas posiciones de una cadena mediante los métodos `charAt()` y `charCodeAt()`, como en este ejemplo:

```
var str:String = "hello world!";
for (var i:int = 0; i < str.length; i++)
{
    trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

Cuando se ejecuta este código, se produce el siguiente resultado:

```
h - 104
e - 101
l - 108
l - 108
o - 111
- 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

También se pueden utilizar códigos de caracteres para definir una cadena con el método `fromCharCode()`, como se muestra en el siguiente ejemplo:

```
var myStr:String = String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
        // Sets myStr to "hello world!"
```

Comparar cadenas

Se pueden utilizar los siguientes operadores para comparar cadenas: `<`, `<=`, `!=`, `==`, `=>` y `>`. Estos operadores se pueden utilizar con sentencias condicionales, como `if` y `while`, como se indica en el siguiente ejemplo:

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
    trace("A < a, B < b, C < c, ...");
}
```

Al usar estos operadores con cadenas, ActionScript considera el valor del código de carácter de cada carácter de la cadena, comparando los caracteres de izquierda a derecha, como se muestra a continuación:

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

Los operadores `==` y `!=` se utilizan para comparar cadenas entre sí y para comparar cadenas con otros tipos de objetos, como se muestra en el siguiente ejemplo:

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

Obtención de representaciones de cadena de otros objetos

Se puede obtener una representación de cadena para cualquier tipo de objeto. Todos los objetos tienen un método `toString()` para este fin:

```
var n:Number = 99.47;
var str:String = n.toString();
    // str == "99.47"
```

Al utilizar el operador de concatenación + con una combinación de objetos String y objetos que no son cadenas, no es necesario utilizar el método `toString()`. Para ver más detalles sobre la concatenación, consulte la siguiente sección.

La función global `String()` devuelve el mismo valor para un objeto determinado como el valor devuelto por el objeto llamando al método `toString()`.

Concatenación de cadenas

La concatenación de cadenas consiste en la unión secuencial de dos cadenas en una sola. Por ejemplo, se puede utilizar el operador suma + para concatenar dos cadenas:

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

También se puede utilizar el operador += para producir el mismo resultado, como se indica en el siguiente ejemplo:

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

Además, la clase String incluye un método `concat()`, que se puede utilizar como se muestra a continuación:

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

Si se utiliza el operador + (o el operador +=) con un objeto String y un objeto que *no* es una cadena, ActionScript convierte automáticamente el objeto que no es una cadena en un objeto String para evaluar la expresión, como se indica en este ejemplo:

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

No obstante, se pueden utilizar paréntesis para agrupar con el fin de proporcionar contexto para el operador +, como se indica en el siguiente ejemplo:

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

Búsqueda de subcadenas y patrones en cadenas

Las subcadenas son caracteres secuenciales dentro de una cadena. Por ejemplo, la cadena `"abc"` tiene las siguientes subcadenas: `""`, `"a"`, `"ab"`, `"abc"`, `"b"`, `"bc"`, `"c"`. ActionScript dispone de métodos para buscar subcadenas de una cadena.

En ActionScript los patrones se definen mediante cadenas o expresiones regulares. Por ejemplo, la siguiente expresión regular define un patrón específico que consiste en las letras A, B y C seguidas de un dígito (las barras diagonales son delimitadores de expresiones regulares):

```
/ABC\d/
```

ActionScript incluye métodos para buscar patrones en cadenas y para reemplazar las coincidencias por subcadenas. Estos métodos se describen en las secciones siguientes.

Las expresiones regulares permiten definir patrones complejos. Para obtener más información, consulte "Utilización de expresiones regulares" en la página 211.

Búsqueda de una subcadena por posición de caracteres

Los métodos `substr()` y `substring()` son similares. Los dos métodos devuelven una subcadena de una cadena. Ambos utilizan dos parámetros. En cada uno de estos métodos, el primer parámetro es la posición del carácter inicial de la cadena en cuestión. No obstante, en el método `substr()`, el segundo parámetro es la *longitud* de la subcadena que debe devolverse, mientras que en el método `substring()`, el segundo parámetro es la posición del carácter *final* de la subcadena (que no se incluye en la cadena devuelta). Este ejemplo muestra la diferencia entre estos dos métodos:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // output: Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

El método `slice()` funciona de forma similar al método `substring()`. Cuando se le facilitan como parámetros dos enteros no negativos, funciona exactamente de la misma forma. No obstante, el método `slice()` admite enteros negativos como parámetros, en cuyo caso la posición del carácter se mide desde el final de la cadena, como se indica en el siguiente ejemplo:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // output: Pari
trace(str.slice(-3,-1)); // output: !!
trace(str.slice(-3,26)); // output: !!!
trace(str.slice(-3,str.length)); // output: !!!
trace(str.slice(-8,-3)); // output: Texas
```

Se pueden combinar enteros positivos y negativos como parámetros del método `slice()`.

Búsqueda de la posición de carácter de una subcadena coincidente

Se pueden utilizar los métodos `indexOf()` y `lastIndexOf()` para localizar subcadenas coincidentes dentro de una cadena, como se muestra en el siguiente ejemplo.

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // output: 10
```

Hay que tener en cuenta que el método `indexOf()` distingue mayúsculas de minúsculas.

Se puede especificar un segundo parámetro para indicar la posición del índice en la cadena desde la que se debe iniciar la búsqueda, de la manera siguiente:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // output: 21
```

El método `lastIndexOf()` localiza la última instancia de una subcadena en la cadena:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // output: 30
```

Si se incluye un segundo parámetro con el método `lastIndexOf()`, la búsqueda se realiza desde esa posición de índice en la cadena hacia atrás (de derecha a izquierda):

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // output: 21
```

Creación de un conjunto de subcadenas segmentadas por un delimitador

Se puede utilizar el método `split()` para crear un conjunto de subcadenas, que se divide en función de un delimitador. Por ejemplo, se puede segmentar en varias cadenas una cadena delimitada por comas o tabulaciones.

En el siguiente ejemplo se muestra la manera de dividir un conjunto en subcadenas con el carácter ampersand (&) como delimitador:

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2); // params == ["first=joe","last=cheng"]
```

El segundo parámetro del método `split()`, que es opcional, define el tamaño máximo del conjunto devuelto.

También se puede utilizar una expresión regular como carácter delimitador:

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

Para más información, consulte "Utilización de expresiones regulares" en la página 211 y Referencia del lenguaje y componentes ActionScript 3.0.

Búsqueda de patrones en cadenas y sustitución de subcadenas

La clase String incluye los siguientes métodos para trabajar con patrones en cadenas:

- Los métodos `match()` y `search()` se utilizan para localizar subcadenas que coincidan con un patrón.
- El método `replace()` permite buscar subcadenas que coincidan con un patrón y sustituirlas por una subcadena especificada.

Estos métodos se describen en las secciones siguientes.

Se pueden utilizar cadenas o expresiones regulares para definir los patrones utilizados en estos métodos. Para obtener más información sobre las expresiones regulares, consulte "Utilización de expresiones regulares" en la página 211.

Búsqueda de subcadenas coincidentes

El método `search()` devuelve la posición del índice de la primera subcadena que coincide con un patrón determinado, como se indica en este ejemplo:

```
var str:String = "The more the merrier.";
// (This search is case-sensitive.)
trace(str.search("the")); // output: 9
```

También se pueden utilizar expresiones regulares para definir el patrón que se debe buscar, como se indica en este ejemplo:

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

La salida del método `trace()` es 0 porque el primer carácter de la cadena es la posición de índice 0. La búsqueda no distingue mayúsculas de minúsculas porque está establecido el indicador `i` en la expresión regular.

El método `search()` busca una sola coincidencia y devuelve su posición de índice inicial, aunque el indicador `g` (global) esté establecido en la expresión regular.

En el siguiente ejemplo se muestra una expresión regular más compleja, que coincide con una cadena entre comillas dobles:

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // output: 4

str = "The \"more the merrier.";
trace(str.search(pattern)); // output: -1
// (Indicates no match, since there is no closing double quotation mark.)
```

El método `match()` funciona de manera similar. Busca una subcadena coincidente. Sin embargo, al utilizar el indicador global en un patrón de expresión regular, como en el siguiente ejemplo, `match()` devuelve un conjunto de subcadenas coincidentes:

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

El conjunto `results` se establece en:

```
["bob@example.com","omar@example.org"]
```

Para obtener más información sobre las expresiones regulares, consulte “Utilización de expresiones regulares” en la página 211“Utilización de expresiones regulares” en la página 211.

Sustitución de subcadenas coincidentes

Se puede utilizar el método `replace()` para buscar un patrón específico en una cadena y sustituir las coincidencias por la cadena de sustitución especificada, como se indica en el siguiente ejemplo:

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch"));
    //sche sells seaschells by the seaschore.
```

En este ejemplo se puede observar que las cadenas coincidentes no distinguen mayúsculas de minúsculas porque está establecido el indicador `i` (`ignoreCase`) en la expresión regular y se sustituyen todas las coincidencias porque está establecido el indicador `g` (`global`). Para obtener más información, consulte “Utilización de expresiones regulares” en la página 211.

Se pueden incluir los siguientes códigos de sustitución `$`en la cadena de sustitución. El texto de sustitución mostrado en la tabla siguiente se inserta en lugar del código de sustitución `$`:

Código $	Texto de sustitución
$$	$
$&	La subcadena coincidente.
$`	La parte de la cadena que precede a la subcadena coincidente. Este código utiliza el carácter de comilla simple recta izquierda (`), no la comilla simple recta (') ni la comilla simple curva izquierda (').
$'	La parte de la cadena que sigue a la subcadena coincidente. Este código utiliza la comilla simple recta (').
$*n*	El *n*-ésimo grupo entre paréntesis coincidente capturado, donde n es un único dígito, 1-9, y $n no va seguido de un dígito decimal.
$*nn*	La *nn*-ésima coincidencia de grupo entre paréntesis capturada, donde *nn* es un número decimal de dos dígitos, 01–99. Si la *nn*-ésima captura no está definida, el texto de sustitución será una cadena vacía.

Por ejemplo, a continuación se muestra el uso de los códigos de sustitución `$2` y `$1`, que representan el primer y el segundo grupo coincidente captado:

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

También se puede utilizar una función como segundo parámetro del método `replace()`. El texto coincidente se sustituye por el valor devuelto por la función.

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));

function usdToEuro(matchedSubstring:String,  capturedMatch1:String,  index:int,
str:String):String
{
    var usd:String = capturedMatch1;
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.853690;
    var euro:Number = parseFloat(usd) * exchangeRate;
    const euroSymbol:String = String.fromCharCode(8364);
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Si se utiliza una función como segundo parámetro del método `replace()`, se pasan los siguientes argumentos a la función:

- La parte coincidente de la cadena.
- Las coincidencias de grupos de paréntesis de captura. El número de argumentos pasados de esta forma varía en función del número de grupos entre paréntesis coincidentes. Se puede determinar el número de grupos entre paréntesis coincidentes comprobando `arguments.length - 3` dentro del código de la función.
- La posición de índice en la que comienza la coincidencia en la cadena.
- La cadena completa.

Conversión de cadenas de mayúsculas a minúsculas y viceversa

Como se indica en el siguiente ejemplo, los métodos `toLowerCase()` y `toUpperCase()` convierten los caracteres alfabéticos de la cadena a minúsculas y a mayúsculas respectivamente.

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

Tras ejecutar estos métodos, la cadena original permanece intacta. Para transformar la cadena original, utilice el código siguiente:

```
str = str.toUpperCase();
```

Estos métodos funcionan con caracteres extendidos, no simplemente a–z y A–Z:

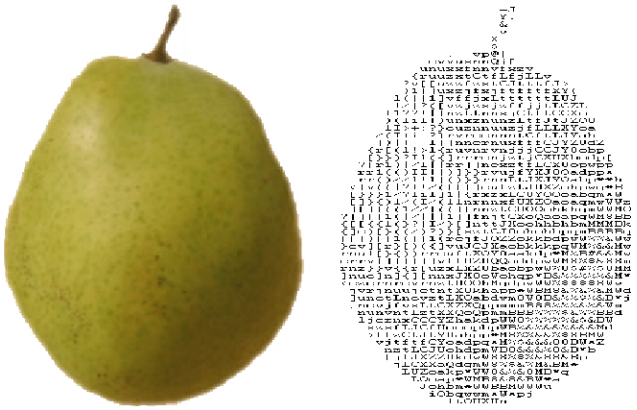
```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

Ejemplo: ASCII art

El ejemplo ASCII Art muestra diversas características del trabajo con la clase String en ActionScript 3.0, como las siguientes:

- Se utiliza el método `split()` de la clase String para extraer valores de una cadena delimitada por caracteres (información de imagen en un archivo de texto delimitado por tabulaciones).
- Se utilizan varias técnicas de manipulación de cadenas, como `split()`, la concatenación y la extracción de una parte de la cadena mediante `substring()` y `substr()` para convertir a mayúsculas la primera letra de cada palabra de los títulos de imagen.
- El método `getCharAt()` se utiliza para obtener un solo carácter de una cadena (a fin de determinar el carácter ASCII correspondiente a un valor de mapa de bits de escala de grises).
- Se utiliza la concatenación de cadenas para generar carácter a carácter la representación ASCII Art de una imagen.

El término *ASCII Art (arte ASCII)* designa representaciones textuales de una imagen, en las que representa la imagen en una cuadrícula de caracteres con fuente de espacio fijo, como los caracteres Courier New. La imagen siguiente muestra un ejemplo de arte ASCII producido por la aplicación:



La versión de arte ASCII del gráfico se muestra a la derecha

Para obtener los archivos de la aplicación para esta muestra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación ASCIIArt se encuentran en la carpeta Samples/AsciiArt. La aplicación consta de los siguientes archivos:

Archivo	Descripción
AsciiArtApp.mxml o AsciiArtApp.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	La clase que proporciona la funcionalidad principal de la aplicación, incluidas la extracción de metadatos de imagen de un archivo de texto, la carga de imágenes y la administración del proceso de conversión de imagen a texto.
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	Una clase que proporciona el método `parseBitmapData()` para convertir datos de imagen en una versión de tipo String.
com/example/programmingas3/asciiArt/Image.as	Una clase que representa una imagen de mapa de bits cargada.
com/example/programmingas3/asciiArt/ImageInfo.as	Una clase que representa metadatos de una imagen de arte ASCII (como el título, el URL del archivo de imagen, etc.).
image/	Una carpeta que contiene imágenes utilizadas por la aplicación.
txt/ImageData.txt	Un archivo de texto delimitado por tabulaciones, que contiene información sobre las imágenes que la aplicación debe cargar.

Extracción de valores delimitados por tabulaciones

En este ejemplo se sigue la práctica común de almacenar datos de aplicación por separado de la aplicación en sí; de esta manera, si los datos cambian (por ejemplo, si se añade otra imagen o cambia el título de una imagen), no es necesario volver a generar el archivo SWF. En este caso, los metadatos de imagen, incluidos el título de la imagen, el URL del archivo de imagen real y algunos valores que se utilizan para manipular la imagen, se almacenan en un archivo de texto (el archivo txt/ImageData.txt file del proyecto). A continuación se describe el contenido del archivo de texto:

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

El archivo utiliza un formato delimitado por tabulaciones específico. La primera línea (fila) es una fila de encabezado. Las restantes líneas contienen los siguientes datos para cada mapa de bits que se va a cargar:

- El nombre de archivo del mapa de bits.
- El nombre para mostrar del mapa de bits.
- Los valores de los umbrales blanco y negro para los mapas de bits. Son valores hexadecimales por encima o por debajo de los cuales, un píxel se considerará completamente blanco o completamente negro.

En cuanto se inicia la aplicación, la clase AsciiArtBuilder carga y analiza el contenido del archivo de texto para crear la "pila" de imágenes que se mostrará, utilizando el código siguiente del método `parseImageInfo()` de la clase AsciiArtBuilder:

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
    var imageInfoRaw:String = lines[i];
    ...
    if (imageInfoRaw.length > 0)
    {
        // Create a new image info record and add it to the array of image info.
        var imageInfo:ImageInfo = new ImageInfo();

        // Split the current line into values (separated by tab (\t)
        // characters) and extract the individual properties:
        var imageProperties:Array = imageInfoRaw.split("\t");
        imageInfo.fileName = imageProperties[0];
        imageInfo.title = normalizeTitle(imageProperties[1]);
        imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
        imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
        result.push(imageInfo);
    }
}
```

Todo el contenido del archivo de texto está en una sola instancia de String, la propiedad `_imageInfoLoader.data`. Si se utiliza el método `split()` con el carácter de nueva línea (`"\n"`) como parámetro, la instancia de String se divide en un conjunto (`lines`) cuyos elementos son las líneas individuales del archivo de texto. A continuación, el código utiliza un bucle para trabajar con cada una de las líneas (salvo la primera, ya que contiene sólo encabezados, no contenido real). Dentro del bucle, se vuelve a utilizar el método `split()` para dividir el contenido de la línea individual en un conjunto de valores (el objeto Array denominado `imageProperties`). El parámetro utilizado con el método `split()` en este caso es el carácter de tabulación (`"\t"`), ya que los valores de cada línea están delimitados por tabulaciones.

Utilización de métodos String para normalizar títulos de imágenes

Una de las decisiones de diseño para esta aplicación es que todos los títulos de imágenes deben mostrarse con un formato estándar, con la primera letra de cada palabra convertida a mayúsculas (con la excepción de unas pocas palabras que no se suelen escribir en mayúsculas en los títulos en inglés). En lugar de suponer que el archivo de texto contiene títulos con el formato correcto, la aplicación aplica el formato a los títulos cuando los extrae del archivo de texto.

En el listado de código anterior se utiliza la siguiente línea de código como parte de la extracción de valores de metadatos de imagen individuales:

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

En ese código, se aplica al título de imagen del archivo de texto el método `normalizeTitle()` antes de almacenarlo en el objeto ImageInfo:

```
private function normalizeTitle(title:String):String
{
    var words:Array = title.split(" ");
    var len:uint = words.length;
    for (var i:uint; i < len; i++)
    {
        words[i] = capitalizeFirstLetter(words[i]);
    }

    return words.join(" ");
}
```

Este método utiliza el método `split()` para dividir el título en palabras individuales (separadas por el carácter espacio), aplica a cada palabra el método `capitalizeFirstLetter()` y después utiliza el método `join()` de la clase Array para volver a combinar las palabras en una sola cadena.

Como indica su nombre, el método `capitalizeFirstLetter()` convierte a mayúscula la primera letra de cada palabra:

```
    /**
     * Capitalizes the first letter of a single word, unless it's one of
     * a set of words that are normally not capitalized in English.
     */
    private function capitalizeFirstLetter(word:String):String
    {
        switch (word)
        {
            case "and":
            case "the":
            case "in":
            case "an":
            case "or":
            case "at":
            case "of":
            case "a":
                // Don't do anything to these words.
                break;
            default:
                // For any other word, capitalize the first character.
                var firstLetter:String = word.substr(0, 1);
                firstLetter = firstLetter.toUpperCase();
                var otherLetters:String = word.substring(1);
                word = firstLetter + otherLetters;
        }
        return word;
    }
```

En inglés, el carácter inicial de cada una de las palabras de un título *no* va en mayúsculas si es alguna de estas palabras: “and”, “the”, “in”, “an”, “or”, “at”, “of”, o “a”. (Ésta es una versión simplificada de las reglas.) Para ejecutar esta lógica, el código utiliza primero una sentencia `switch` para comprobar si la palabra es una de las palabras que no se deben escribir en mayúsculas. Si es así, el código simplemente sale de la sentencia `switch`. Por otra parte, si hubiera que escribir la palabra en mayúsculas, se hará en varios pasos, como se indica a continuación:

1. La primera letra de la palabra se extrae mediante `substr(0, 1)`, que extrae una subcadena que empieza por el carácter correspondiente al índice 0 (la primera letra de la cadena, como indica el primer parámetro `0`). La subcadena tendrá una longitud de un carácter (como indica el segundo parámetro, `1`).
2. El carácter se convierte a mayúscula mediante el método `toUpperCase()`.
3. Los caracteres restantes de la palabra original se extraen mediante `substring(1)`, que extrae una subcadena que empieza en el índice 1 (la segunda letra) hasta el final de la cadena (lo que se indica omitiendo el segundo parámetro del método `substring()`).
4. La última palabra se crea combinando la primera letra que se acaba de convertir en mayúscula con las restantes letras mediante concatenación de cadenas: `firstLetter + otherLetters`.

Creación de texto de arte ASCII

La clase BitmapToAsciiConverter proporciona la funcionalidad de convertir una imagen de mapa de bits en su representación de texto ASCII. Este proceso, que se muestra aquí parcialmente, lo lleva a cabo el método `parseBitmapData()`:

```
    var result:String = "";

    // Loop through the rows of pixels top to bottom:
    for (var y:uint = 0; y < _data.height; y += verticalResolution)
    {
        // Within each row, loop through pixels left to right:
        for (var x:uint = 0; x < _data.width; x += horizontalResolution)
        {
            ...

            // Convert the gray value in the 0-255 range to a value
            // in the 0-64 range (since that's the number of "shades of
            // gray" in the set of available characters):
            index = Math.floor(grayVal / 4);
            result += palette.charAt(index);
        }
        result += "\n";
    }
    return result;
```

Este código define primero una instancia de String denominada `result` que se utilizará para generar la versión de arte ASCII de la imagen de mapa de bits. A continuación, recorre píxeles individuales de la imagen de mapa de bits original. Utiliza varias técnicas de manipulación de colores (que se omiten aquí por brevedad), convierte los valores de los colores rojo, verde y azul de un píxel individual en un solo valor de escala de grises (un número entre 0 y 255). A continuación el código divide ese valor por 4 (como se indica) para convertirlo en un valor en la escala 0-63, que se almacena en la variable `index`. (Se utiliza la escala 0-63 porque la "paleta" de caracteres ASCII disponibles que utiliza esta aplicación contiene 64 valores.) La paleta de caracteres se define como una instancia de String en la clase BitmapToAsciiConverter:

```
// The characters are in order from darkest to lightest, so that their
// position (index) in the string corresponds to a relative color value
// (0 = black).
private static const palette:String =
"@#$%&8BMW*mwqpdbkhaoQ0OZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

Como la variable `index` define el carácter ASCII de la paleta que corresponde al píxel actual de la imagen de mapa de bits, ese carácter se recupera de la instancia String de la `paleta` mediante el método `charAt()`. A continuación se añade `result` a la instancia de tipo String mediante el operador de concatenación y asignación (`+=`). Además, al final de cada fila de píxeles, un carácter de nueva línea se concatena con el final de la cadena`result`, lo que fuerza un ajuste de línea para crear una nueva fila de "píxeles" de caracteres.

Capítulo 8: Trabajo con conjuntos

Los conjuntos permiten almacenar varios valores en una sola estructura de datos. Se pueden utilizar conjuntos indexados simples que almacenan valores con índices enteros ordinales fijos o conjuntos asociativos complejos que almacenan valores con claves arbitrarias. Los conjuntos también pueden ser multidimensionales y contener elementos que son conjuntos en sí mismos. Finalmente, puede utilizar un vector para un conjunto cuyos elementos son todos instancias del mismo tipo de datos. En este capítulo se explica la manera de crear y manipular diversos tipos de conjuntos.

Fundamentos de la utilización de conjuntos

Introducción a la utilización de conjuntos

Al programar es frecuente tener que trabajar con un conjunto de elementos en lugar de con un solo objeto. Por ejemplo, una aplicación de reproductor de música puede tener una lista de canciones esperando para ser reproducidas. No tiene sentido crear una variable independiente para cada canción de la lista. Es preferible agrupar todos los objetos Song y trabajar con ellos como un grupo.

Un conjunto es un elemento de programación que actúa como un contenedor para un grupo de elementos, como una lista de canciones. Normalmente todos los elementos de un conjunto son instancias de la misma clase, pero esto no es un requisito en ActionScript. Los elementos individuales de un conjunto se denominan *elementos* del conjunto. Un conjunto es como un cajón de archivador para variables. Se pueden añadir variables al conjunto como elementos (es como colocar una carpeta en el cajón del archivador). Se puede trabajar con el conjunto como una sola variable (es como llevar todo el cajón a un lugar distinto). Se puede trabajar con las variables como un grupo (es como recorrer las carpetas una a una buscando información). También se puede acceder a ellas de forma individual (es como abrir el cajón y seleccionar una sola carpeta).

Por ejemplo, suponga que está creando una aplicación de reproductor de música en la que un usuario puede seleccionar varias canciones y añadirlas a la lista de reproducción. Se puede crear en el código ActionScript un método denominado `addSongsToPlaylist()` que acepte un solo conjunto como parámetro. Independientemente del número de canciones que se añada a la lista (unas pocas, muchas o sólo una), basta con llamar al método `addSongsToPlaylist()` una vez para transferirle el conjunto que contiene los objetos Song. Se puede utilizar un bucle en el método `addSongsToPlaylist()` para recorrer los elementos del conjunto (las canciones) de uno en uno y añadirlos a la lista de reproducción.

El tipo más común de conjunto de ActionScript es un *conjunto indexado*. En un conjunto indexado, cada elemento se almacena en una posición numerada (que recibe el nombre de *índice*). Para acceder a los elementos, se utiliza el número, como en una dirección. Los conjuntos indexados son suficiente para la mayoría de las necesidades de programación. La clase Array es una de las clases habituales utilizadas para representar un conjunto indexado.

A menudo, se utiliza un conjunto indexado para guardar varios elementos del mismo tipo (objetos que son instancias de la misma clase). La clase Array no tiene capacidad para restringir el tipo de elementos que contiene. La clase Vector es un tipo de conjunto indexado en el que todos los elementos de un conjunto simple son del mismo tipo. Al utilizar una instancia de Vector en lugar de una instancia de Array, también se consiguen mejoras de rendimiento y otras ventajas. la clase Vector está disponible a partir de Flash Player 10 y Adobe AIR 1.5.

Un uso especial de un conjunto indexado es un *conjunto multidimensional*. Un conjunto multidimensional es un conjunto indexado cuyos elementos son conjuntos indexados (que, a su vez, contienen otros elementos).

Otro tipo de conjunto es el *conjunto asociativo*, que utiliza una *clave* de tipo String en lugar de un índice numérico para identificar elementos individuales. Por último, ActionScript 3.0 también contiene la clase Dictionary, que representa un *diccionario*. Un diccionario es un conjunto que permite utilizar cualquier tipo de objeto como clave para distinguir entre elementos.

Tareas comunes con conjuntos

En este capítulo se describen las siguientes actividades comunes para trabajar con conjuntos:

- Crear conjuntos indexados con la clase Array y la clase Vector
- Añadir y eliminar elementos de conjunto
- Ordenar elementos de conjunto
- Extraer partes de un conjunto
- Trabajo con conjuntos asociativos y diccionarios
- Trabajo con conjuntos multidimensionales
- Copiar elementos de conjunto
- Crear una subclase de conjunto

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Conjunto: objeto que se utiliza como contenedor para agrupar varios objetos.
- Operador de acceso a conjunto (`[]`): par de corchetes que encierran un índice o una clave que identifica de forma exclusiva a un elemento de conjunto. Esta sintaxis se utiliza después de un nombre de variable de conjunto para especificar un solo elemento del conjunto, no el conjunto completo.
- Conjunto asociativo: conjunto que utiliza claves de tipo cadena para identificar elementos individuales.
- Tipo base: tipo de datos que puede almacenar una instancia de Vector.
- Diccionario: conjunto cuyos elementos constan de un par de objetos, denominados clave y valor. Se utiliza la clave en lugar de un índice numérico para identificar un elemento individual.
- Elemento: elemento individual de un conjunto.
- Índice: "dirección numérica" que se utiliza para identificar un elemento individual de un conjunto indexado.
- Conjunto indexado: tipo estándar de conjunto que almacena cada elemento en una posición numerada y utiliza el número (índice) para identificar elementos individuales.
- Clave: cadena u objeto que se utiliza para identificar un elemento individual en un conjunto asociativo o un diccionario.
- Conjunto multidimensional: conjunto que contiene elementos que son conjuntos en lugar de valores individuales.
- `T`: convención estándar que se utiliza en esta documentación para representar el tipo base de una instancia de Vector, independientemente del tipo base. La convención T se utiliza para representar un nombre de clase, tal como se indica en la descripción del parámetro Type. ("T" viene de "tipo", como en "tipo de datos".)
- Parámetro Type: sintaxis que se utiliza en el nombre de la clase Vector para especificar el tipo base del vector (tipo de datos de los objetos que almacena). La sintaxis está formada por un punto (`.`) seguido del nombre del tipo de datos encerrado entre paréntesis angulares (`<>`). Todo junto, sería algo así: `Vector.<T>`. En esta documentación, la clase especificada en el parámetro Type se representa genéricamente como `T`.

- Vector: tipo de conjunto cuyos elementos son todos instancias del mismo tipo de datos.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Prácticamente todos los listados de código de este capítulo incluyen la llamada a la función `trace()` apropiada. Para probar los listados de código de este capítulo:

1. Cree un documento vacío utilizando la herramienta de edición de Flash.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Ejecute el programa seleccionando Control > Probar película.

   El resultado de la función `trace()` se ve en el panel Salida.

Ésta y otras técnicas para probar los listados de código de ejemplo se describen de forma detallada en "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

Conjuntos indexados

Los conjuntos indexados almacenan una serie de uno o varios valores organizados de forma que se pueda acceder a cada valor mediante un valor entero sin signo. El primer número de índice siempre es 0 y aumenta una unidad por cada elemento que se añada al conjunto. En ActionScript 3.0, se utilizan dos clases como conjuntos indexados: la clase Array y la clase Vector.

Los conjuntos indexados usan un entero de 32 bits sin signo como número de índice. El tamaño máximo de un conjunto indexado es 2^{32} - 1 ó 4.294.967.295. Si se intenta crear un conjunto de tamaño superior al tamaño máximo, se producirá un error en tiempo de ejecución.

Para acceder a un elemento individual de un conjunto indexado, debe utilizarse el operador de acceso a conjunto (`[]`) para especificar la posición del índice del elemento al que se desee acceder. Por ejemplo, el siguiente código representa el primer elemento (elemento en el índice 0) de un conjunto indexado llamado `songTitles`:

```
songTitles[0]
```

La combinación del nombre de la variable de conjunto seguido del índice entre corchetes hace las veces de identificador único. (Dicho de otro modo, se puede utilizar en cualquier situación en la que se utilizaría un nombre de variable). Es posible asignar un valor a un elemento de conjunto indexado utilizando el nombre y el índice situados a la izquierda de la sentencia de asignación:

```
songTitles[1] = "Symphony No. 5 in D minor";
```

Del mismo modo, se puede recuperar el valor de un elemento de conjunto indexado utilizando el nombre y el índice situados a la derecha de una sentencia de asignación:

```
var nextSong:String = songTitles[2];
```

También se puede utilizar una variable entre corchetes, en vez de proporcionar un valor explícito. (La variable debe contener un valor entero no negativo, como un uint, un int positivo o una instancia positiva entera de Number). Esta técnica suele utilizarse para "recorrer en bucle" los elementos de un conjunto indexado y llevar a cabo una operación en uno o en todos los elementos. El siguiente código muestra esta técnica. El código utiliza un bucle para acceder a cada valor de un objeto Array llamado `oddNumbers`. Utiliza la sentencia `trace()` para imprimir cada valor con la forma "oddNumber[*índice*] = *valor*":

```
var oddNumbers:Array = [1, 3, 5, 7, 9, 11];
var len:uint = oddNumbers.length;
for (var i:uint = 0; i < len; i++)
{
    trace("oddNumbers[" + i.toString() + "] = " + oddNumbers[i].toString());
}
```

Clase Array

El primer tipo de conjunto indexado es la clase Array. Una instancia de Array puede contener un valor de cualquier tipo de datos. El mismo objeto Array puede contener objetos que sean de distinto tipo de datos. Por ejemplo, una sola instancia de Array puede tener un valor de String en el índice 0, una instancia de Number en el índice 1 y un objeto XML en el índice 2.

Clase Vector

Otro tipo de conjunto indexado disponible en ActionScript 3.0 es la clase Vector. Una instancia de Vector es un *conjunto de tipos*, lo que significa que todos los elementos de una instancia de Vector siempre son del mismo tipo.

Nota: *la clase Vector está disponible a partir de Flash Player 10 y Adobe AIR 1.5.*

Al declarar una variable Vector o crear una instancia de un objeto Vector, se especifica explícitamente el tipo de datos de los objetos que dicho vector puede contener. El tipo de datos especificado se conoce como *tipo base* del vector. En tiempo de ejecución y de compilación (en modo estricto), se comprueba cualquier código que establezca el valor de un elemento Vector o que recupere un valor desde un vector. Si el tipo de datos del objeto que se añade o se recupera no coincide con el tipo base del vector, se produce un error.

Además de la restricción del tipo de datos, la clase Vector tiene otras limitaciones que la distinguen de la clase Array:

- Una vector es un conjunto denso. Un objeto Array puede tener valores en los índices 0 y 7, incluso si no tiene valores en las posiciones 1-6. Sin embargo, un vector debe tener un valor (o `null`) en todos los índices.
- Opcionalmente, un vector puede tener longitud fija. Esto significa que el número de elementos contenidos por el vector no puede cambiar.
- El acceso a los elementos de un vector está definido por sus límites. Nunca se puede leer un valor de un índice superior a la (`longitud` - 1) del elemento final. Nunca se puede establecer un valor con un índice superior al índice final actual. Dicho de otro modo, sólo se puede establecer un valor en un índice existente o en una `[longitud]` de índice.

La consecuencia de estas restricciones hace que utilizar un vector presente dos ventajas frente a una instancia de Array cuyos elementos sean todos instancias de una sola clase:

- Rendimiento: el acceso y la iteración en los elementos de un conjunto son mucho más rápidos si se utiliza una instancia de Vector y no una de Array.
- Seguridad de tipos: en modo estricto, el compilador puede identificar errores de tipos de datos. Algunos ejemplos de este tipo de errores incluyen la asignación de un valor del tipo de datos incorrecto a un vector, o esperar un tipo de datos incorrecto al leer un valor en un vector. En tiempo de ejecución, los tipos de datos también se comprueban al añadir datos o leerlos en un objeto Vector. Tenga en cuenta, no obstante, que cuando se utiliza el método `push()` o el método `unshift()` para añadir valores a un vector, los tipos de datos de los argumentos no se comprueban en tiempo de compilación. Cuando se utilizan estos métodos, los valores se siguen comprobando en tiempo de ejecución.

Dejando a un lado las restricciones y ventajas adicionales, la clase Vector es muy similar a la clase Array. Las propiedades y los métodos de un objeto Vector son similares (la mayoría de las veces, idénticos) a las propiedades y métodos de un objeto Array. En todas las situaciones en las que utilizaría un objeto Array con todos sus elementos del mismo tipo de datos, es preferible usar una instancia del objeto Vector.

Creación de conjuntos

Se pueden utilizar diversas técnicas para crear una instancia de Array o una instancia de Vector. No obstante, las técnicas para crear cada tipo de conjunto son ligeramente distintas.

Creación de una instancia de Array

Para crear un objeto Array, llame al constructor `Array()` o utilice la sintaxis literal de Array.

La función constructora de `Array()` se puede utilizar de tres maneras distintas. En primer lugar, si se llama al constructor sin argumentos, se obtiene un conjunto vacío. Se puede utilizar la propiedad `length` de la clase Array para comprobar que el conjunto no tiene elementos. Por ejemplo, el código siguiente llama al constructor de `Array()` sin argumentos:

```
var names:Array = new Array();
trace(names.length); // output: 0
```

En segundo lugar, si se utiliza un número como único parámetro del constructor de `Array()`, se crea un conjunto de esa longitud y se establece el valor de cada elemento en `undefined`. El argumento debe ser un entero sin signo entre 0 y 4.294.967.295. Por ejemplo, el código siguiente llama al constructor de `Array()` con un solo argumento numérico:

```
var names:Array = new Array(3);
trace(names.length); // output: 3
trace(names[0]); // output: undefined
trace(names[1]); // output: undefined
trace(names[2]); // output: undefined
```

En tercer lugar, si se llama al constructor y se le pasa una lista de elementos como parámetros, se crea un conjunto con elementos correspondientes a cada uno de los parámetros. El código siguiente pasa tres argumentos al constructor de `Array()`:

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // output: 3
trace(names[0]); // output: John
trace(names[1]); // output: Jane
trace(names[2]); // output: David
```

También es posible crear conjuntos con literales de conjunto. Un literal de conjunto se puede asignar directamente a una variable de tipo conjunto, como se indica en el siguiente ejemplo:

```
var names:Array = ["John", "Jane", "David"];
```

Creación de una instancia de Vector

Para crear una instancia de Vector, llame al constructor `Vector.<T>()`. También es posible crear un vector llamando a la función global `Vector.<T>()`. Esta función convierte el objeto especificado en una instancia de Vector. ActionScript no tiene ningún equivalente de la sintaxis de literal de Array para el elemento Vector.

Cada vez que se declara una variable Vector (o, del mismo modo, un parámetro del método Vector o un tipo devuelto del método), se debe especificar el tipo base de la variable Vector. También se debe especificar el tipo base al crear una instancia de Vector llamando al constructor `Vector.<T>()`. Dicho de otro modo, cada vez que se utiliza el término `Vector` en ActionScript, va acompañado de un tipo base.

Puede especificar el tipo base de Vector mediante la sintaxis de parámetros de tipos. El parámetro type va inmediatamente después de la palabra `Vector` en el código. Está formado por un punto (`.`) seguido de la clase base encerrada entre paréntesis angulares (`<>`), tal como puede verse en este ejemplo:

```
var v:Vector.<String>;
v = new Vector.<String>();
```

En la primera línea del ejemplo, la variable `v` se declara como una instancia de `Vector.<String>`. Dicho de otro modo, representa un conjunto indexado que sólo puede contener instancias de String. La segunda línea llama al constructor `Vector()` para crear una instancia del mismo tipo de vector (es decir, un vector cuyos elementos sean todos objetos String). Asigna el objeto a `v`.

Utilización del constructor Vector.<T>()

Si utiliza el constructor `Vector.<T>()` sin ningún argumento, crea una instancia vacía de Vector. Puede comprobar si un vector está vacío observando su propiedad `length`. Por ejemplo, el siguiente código llama al constructor `Vector.<T>()` sin ningún argumento:

```
var names:Vector.<String> = new Vector.<String>();
trace(names.length); // output: 0
```

Si sabe con antelación el número de elementos que necesita el vector inicialmente, puede predefinir el número de elementos contenidos en Vector. Para crear un vector con un determinado número de elementos, transfiera el número de elementos como primer parámetro (el parámetro `length`). Puesto que los elementos Vector no pueden estar vacíos, se llenan con instancias del tipo base. Si el tipo base es un tipo de referencia que admite valores `null`, todos los elementos contendrán `null`. En caso contrario, todos los elementos contienen el valor predeterminado para la clase. Por ejemplo, una variable uint no puede ser `null`. En consecuencia, en el siguiente código, el vector llamado `ages` se crea con siete elementos (cada uno con el valor 0):

```
var ages:Vector.<uint> = new Vector.<uint>(7);
trace(ages); // output: 0,0,0,0,0,0,0
```

Finalmente, con el constructor `Vector.<T>()`, también es posible crear un vector de longitud fija si se transfiere `true` como segundo parámetro (el parámetro `fixed`). En ese caso, el vector se crea con el número especificado de elementos y este número no puede cambiar. Debe tenerse en cuenta, no obstante, que sigue siendo posible cambiar los valores de los elementos de un vector de longitud fija.

Al contrario de lo que sucede con la clase Array, no es posible transferir una lista de valores al constructor `Vector.<T>()` para especificar los valores iniciales del vector.

Utilización de la función global Vector.<T>()

Además del constructor `Vector.<T>()`, se puede utilizar también la función global `Vector.<T>()` para crear un objeto Vector. La función global `Vector.<T>()` es una función de conversión. Al llamar a la función global `Vector.<T>()`, se especifica el tipo base del vector devuelto por el método. Se transfiere un conjunto indexado sencillo (instancia de Array o de Vector) como argumento. El método devuelve un vector con el tipo base especificado y con los valores del argumento del conjunto original. El siguiente código muestra la sintaxis para llamar a la función global `Vector.<T>()`:

```
var friends:Vector.<String> = Vector.<String>(["Bob", "Larry", "Sarah"]);
```

La función global `Vector.<T>()` lleva a cabo la conversión de tipos de datos en dos niveles. En primer lugar, cuando se transfiere una instancia de Array a la función, se devuelve una instancia de Vector. En segundo lugar, sea o no el conjunto original una instancia de Array o de Vector, la función intenta convertir los elementos del conjunto original en valores del tipo base. La conversión utilizar reglas de conversión estándar de tipos de datos de ActionScript. Por ejemplo, el siguiente código convierte los valores de String del objeto Array original en enteros del vector resultante. La parte decimal del primer valor (`"1.5"`) se corta y el tercer valor no numérico (`"Waffles"`) se convierte a 0 en el resultado:

```
var numbers:Vector.<int> = Vector.<int>("1.5", "17", "Waffles"]);
trace(numbers); // output: 1,17,0
```

Si no es posible convertir alguno de los elementos originales, se produce un error.

Cuando el código llama a la función global `Vector.<T>()`, si un elemento del conjunto original es una instancia de una subclase del tipo base especificado, el elemento se añade al vector resultante (y no se produce ningún error). Utilizar la función global `Vector.<T>()` es el único modo de convertir un vector con tipo base `T` en uno con un tipo base que sea una superclase de `T`.

Inserción de elementos de conjunto

La forma más sencilla de añadir un elemento a un conjunto indexado es utilizar el operador de acceso a conjunto (`[]`). Para establecer el valor de un elemento de conjunto indexado, utilice el nombre del objeto Array o Vector y el número de índice situado a la izquierda de una sentencia de asignación:

```
songTitles[5] = "Happy Birthday";
```

Si el objeto Array o Vector no tiene aún ningún elemento en ese índice, éste se crea y se almacena el valor en él. Si ya existe un valor en el índice, el nuevo valor sustituye al existente.

Un objeto Array permite crear un elemento en cualquier índice. Sin embargo, con un objeto Vector sólo es posible asignar un valor a un índice existente o al siguiente índice disponible. El siguiente índice disponible corresponde a la propiedad `length` del objeto Vector. El modo más seguro de añadir un nuevo elemento a un objeto Vector es utilizar un código como el siguiente:

```
myVector[myVector.length] = valueToAdd;
```

Tres de los métodos de la clase Array y Vector (`push()`, `unshift()` y `splice()`) permiten insertar elementos en un conjunto indexado. El método `push()` añade uno o más elementos al final de un conjunto. Es decir, el último elemento insertado en el conjunto mediante el método `push()` tendrá el número de índice más alto. El método `unshift()` inserta uno o más elementos al principio de un conjunto, que siempre tiene el número de índice 0. El método `splice()` insertará un número arbitrario de elementos en un índice especificado del conjunto.

En el ejemplo siguiente se ilustran los tres métodos. Un conjunto denominado `planets` se crea para almacenar los nombres de los planetas en orden de proximidad al sol. En primer lugar, se llama al método `push()` para agregar el elemento inicial, `Mars`. En segundo lugar, se llama al método `unshift()` para insertar el elemento que debe estar al principio del conjunto, `Mercury`. Por último, se llama al método `splice()` para insertar los elementos `Venus` y `Earth` a continuación de `Mercury`, pero antes de `Mars`. El primer argumento enviado a `splice()`, el entero 1, dirige la inserción para que se realice en el número de índice 1. El segundo argumento enviado a `splice()`, el entero 0, indica que no se debe eliminar ningún elemento. Por último, el tercer y el cuarto argumento enviados a `splice()`, `Venus` y `Earth` son los elementos que se van a insertar.

```
var planets:Array = new Array();
planets.push("Mars"); // array contents: Mars
planets.unshift("Mercury"); // array contents: Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets); // array contents: Mercury,Venus,Earth,Mars
```

Los métodos `push()` y `unshift()` devuelven un entero sin signo que representa la longitud del conjunto modificado. El método `splice()` devuelve un conjunto vacío cuando se utiliza para insertar elementos, algo que puede parecer extraño, pero que tiene sentido si se tiene en cuenta la versatilidad que ofrece dicho método. Se puede utilizar el método `splice()` no sólo para insertar elementos en un conjunto, sino también para eliminar elementos de un conjunto. Cuando se utiliza para eliminar elementos, el método `splice()` devuelve un conjunto que contiene los elementos eliminados.

Nota: *si la propiedad `fixed` de un objeto Vector es `true`, el número total de elementos de dicho vector es invariable. Si intenta añadir un nuevo elemento a un vector de longitud fija con las técnicas descritas anteriormente, se producirá un error.*

Recuperación de valores y eliminación de elementos de conjunto

El modo más sencillo de recuperar el valor de un elemento en un conjunto indexado es utilizar el operador de acceso a conjunto (`[]`). Para recuperar el valor de un elemento de conjunto indexado, utilice el nombre del objeto Array o Vector y el número de índice situado a la derecha de una sentencia de asignación:

```
var myFavoriteSong:String = songTitles[3];
```

Se puede intentar recuperar un valor de un objeto Array o Vector utilizando un índice en el que no exista ningún elemento. En ese caso, un objeto Array devuelve el valor undefined y un objeto Vector emite una excepción RangeError.

Tres métodos de las clases Array y Vector (`pop()`, `shift()` y `splice()`) permiten eliminar elementos. El método `pop()` elimina un elemento del final del conjunto. Es decir, elimina el elemento que tenga el número de índice más alto. El método `shift()` elimina un elemento del principio del conjunto, lo que significa que siempre elimina el elemento con el número de índice 0. El método `splice()`, que también se puede utilizar para insertar elementos, elimina un número arbitrario de elementos empezando por el que tiene el número de índice especificado por el primer argumento enviado al método.

En el ejemplo siguiente se utilizan los tres métodos para eliminar elementos de una instancia de Array. Se crea un conjunto denominado `oceans` para almacenar los nombres de grandes masas de agua. Algunos de los nombres del conjunto son lagos, no océanos, por lo que hay que eliminarlos.

En primer lugar, se utiliza el método `splice()` para eliminar los elementos `Aral` y `Superior`, e insertar los elementos `Atlantic` e `Indian`. El primer argumento enviado a `splice()`, el entero 2, indica que la operación debe empezar por el tercer elemento de la lista, que tiene el índice 2. El segundo argumento, 2, indica que hay que eliminar dos elementos. Los restantes argumentos, `Atlantic` e `Indian`, son valores que deben insertarse a partir del índice 2.

En segundo lugar, se utiliza el método `pop()` para eliminar el último elemento del conjunto, `Huron`. Y por último, se utiliza el método `shift()` para eliminar el primer elemento del conjunto, `Victoria`.

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian", "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // replaces Aral and Superior
oceans.pop(); // removes Huron
oceans.shift(); // removes Victoria
trace(oceans);// output: Pacific,Arctic,Atlantic,Indian
```

Los métodos `pop()` y `shift()` devuelven el elemento eliminado. En una instancia de Array, el tipo de datos del valor devuelto es Object, ya que los conjuntos pueden contener valores de cualquier tipo de datos. En una instancia de Vector, el tipo de datos del valor es el tipo base del vector. El método `splice()` devuelve un objeto Array o Vector que contiene los valores eliminados. Se puede modificar el ejemplo del conjunto `oceans` de forma que la llamada a `splice()` asigne el conjunto devuelto a una nueva variable de Array, como se indica en el siguiente ejemplo:

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // output: Aral,Superior
```

Es posible encontrar código que utilice el operador `delete` en un elemento del objeto Array. El operador `delete` establece el valor de un elemento de conjunto en `undefined`, pero no elimina el elemento del objeto Array. Por ejemplo, el código siguiente utiliza el operador `delete` en el tercer elemento del conjunto `oceans`, pero la longitud del conjunto sigue siendo 5:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);// output: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]); // output: undefined
trace(oceans.length); // output: 5
```

Se puede utilizar la propiedad `length` de un conjunto para truncar un objeto Array o Vector. Si se establece la propiedad `length` de un conjunto indexado como una longitud inferior a la longitud actual del conjunto, éste se trunca y se eliminan todos los valores almacenados en los números de índice superiores al nuevo valor de `length` menos 1. Por ejemplo, si el conjunto `oceans` se ordena de modo que todas las entradas válidas están al principio del conjunto, se puede utilizar la propiedad `length` para eliminar las entradas situadas al final, tal como se indica en el siguiente código:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // output: Arctic,Pacific
```

***Nota:** si la propiedad `fixed` de un objeto Vector es `true`, el número total de elementos de dicho vector es invariable. Si intenta eliminar o truncar un elemento de un vector de longitud fija utilizando las técnicas descritas anteriormente, se producirá un error.*

Ordenación de un conjunto

Hay tres métodos, `reverse()`, `sort()` y `sortOn()`, que permiten cambiar el orden de un conjunto indexado, ordenando o invirtiendo el orden. Todos estos métodos modifican el conjunto existente. En la tabla siguiente se resumen estos métodos y su comportamiento en los objetos Array y Vector:

Método	Comportamiento de Array	Comportamiento de Vector
`reverse()`	Cambia el orden del conjunto de forma que el último elemento se convierte en el primero, el penúltimo elemento se convierte en el segundo, etc.	Idéntico al comportamiento de Array.
`sort()`	Permite ordenar los elementos del conjunto de diversas formas predefinidas, por ejemplo, por orden alfabético o numérico. También es posible especificar un algoritmo de ordenación personalizado.	Ordena los elementos según el algoritmo de ordenación personalizado especificado.
`sortOn()`	Permite ordenar objetos con una o varias propiedades comunes y utilizar las propiedades como claves de ordenación.	No disponible en la clase Vector.

Método reverse()

El método `reverse()` no admite parámetros y no devuelve un valor, pero permite alternar el orden del conjunto de su estado actual al orden inverso. El ejemplo siguiente invierte el orden de los océanos que se muestra en el conjunto `oceans`:

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // output: Pacific,Indian,Atlantic,Arctic
```

Ordenación básica con el método sort() (sólo en la clase Array)

En una instancia de Array, el método `sort()` reorganiza los elementos de un conjunto con el *orden de clasificación predeterminado*. El orden de clasificación predeterminado tiene las siguientes características:

- En la ordenación se distinguen mayúsculas de minúsculas, lo que significa que los caracteres en mayúsculas preceden a los caracteres en minúsculas. Por ejemplo, la letra D precede a la letra b.
- La ordenación es ascendente, lo que significa que los códigos de caracteres inferiores (como A) preceden a los códigos de caracteres superiores (como B).
- La ordenación coloca juntos los valores idénticos, pero no usa un orden específico.

- La ordenación se basa en cadenas, lo que significa que los elementos se convierten en cadenas antes de ser comparados (por ejemplo, 10 precede a 3 porque el código de carácter de la cadena `"1"` es inferior al de la cadena `"3"`).

A veces hay que ordenar el conjunto sin tener en cuenta mayúsculas o minúsculas, o en orden descendente, o si el conjunto contiene números hay que ordenar numéricamente en lugar de alfabéticamente. El método `sort()` de la clase Array tiene un parámetro `options` que permite modificar todas las características del orden de clasificación predeterminado. Las opciones se definen mediante un conjunto de constantes estáticas de la clase Array, como se indica en la lista siguiente:

- `Array.CASEINSENSITIVE`: esta opción hace que no se tengan en cuenta las diferencias de mayúsculas y minúsculas en la ordenación. Por ejemplo, la b minúscula precede a la D mayúscula.
- `Array.DESCENDING`: invierte la ordenación ascendente predeterminada. Por ejemplo, la letra B precede a la letra A.
- `Array.UNIQUESORT`: hace que se cancele la ordenación si se encuentran dos valores idénticos.
- `Array.NUMERIC`: hace que la ordenación sea numérica, de forma que 3 preceda a 10.

En el ejemplo siguiente se ilustran algunas de estas opciones. Se crea un conjunto denominado `poets` que se ordena con varias opciones distintas.

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // default sort
trace(poets); // output: Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // output: Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // output: cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // use two options
trace(poets); // output: Dante,cummings,Blake,Angelou
```

Ordenación personalizada con el método sort() (clases Array y Vector)

Además de la ordenación básica disponible en los objetos Array, también es posible definir una regla de ordenación personalizada. Esta técnica es la única forma del método `sort()` disponible para la clase Vector. Para definir una ordenación personalizada, debe escribir una función de ordenación personalizada y transferirla como argumento al método `sort()`.

Por ejemplo, si se tiene una lista de nombres en la que cada elemento de la lista contiene el nombre completo de una persona, pero se desea ordenar la lista por apellido, hay que utilizar una función de ordenación personalizada que analice cada elemento y use el apellido en la función de ordenación. El código siguiente muestra cómo se puede hacer esto con una función personalizada que se utiliza como parámetro del método `Array.sort()`:

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
    var lastName:RegExp = /\b\S+$/;
    var name1 = a.match(lastName);
    var name2 = b.match(lastName);
    if (name1 < name2)
    {
        return -1;
    }
    else if (name1 > name2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
trace(names); // output: John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // output: Jane Doe,Mike Jones,John Q. Smith
```

La función de ordenación personalizada `orderLastName()` utiliza una expresión regular para extraer el apellido de cada elemento que se va a utilizar para la operación de comparación. Se utiliza el identificador de la función `orderLastName` como único parámetro al llamar al método `sort()` del conjunto `names`. La función de ordenación acepta dos parámetros, `a` y `b`, ya que procesa dos elementos de conjunto cada vez. El valor devuelto por la función de ordenación indica cómo deben ordenarse los elementos:

- Si el valor devuelto es -1, indica que el primer parámetro, `a`, precede al segundo parámetro, `b`.
- Si el valor devuelto es 1, indica que el segundo parámetro, `b`, precede al primero, `a`.
- Si el valor devuelto es 0, indica que los elementos tienen igual precedencia de ordenación.

Método sortOn() (sólo en la clase Array)

El método `sortOn()` está diseñado para objetos Array con elementos que contienen objetos. Se espera que estos objetos tengan al menos una propiedad en común que se pueda utilizar como criterio de ordenación. El uso del método `sortOn()` en conjuntos de cualquier otro tipo puede producir resultados inesperados.

Nota: *la clase Vector no incluye un método `sortOn()`. Este método sólo está disponible en objetos Array.*

El ejemplo siguiente revisa el conjunto `poets` de forma que cada elemento sea un objeto en lugar de una cadena. Cada objeto contiene el apellido del poeta y el año de nacimiento.

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

Se puede utilizar el método `sortOn()` para ordenar el conjunto por la propiedad `born`. El método `sortOn()` define dos parámetros, `fieldName` y `options`. El argumento `fieldName` debe especificarse como una cadena. En el ejemplo siguiente, se llama a `sortOn()` con dos argumentos, `"born"` y `Array.NUMERIC`. Se utiliza el argumento `Array.NUMERIC` para asegurarse de que la ordenación se realiza numéricamente en lugar de alfabéticamente. Esto es una práctica recomendable, incluso en el caso de que todos los números tengan el mismo número de dígitos, ya que garantiza que la ordenación seguirá comportándose de la manera esperada si posteriormente se añade un número con menos (o más) dígitos al conjunto.

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
    trace(poets[i].name, poets[i].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Ordenación sin modificar el conjunto original (sólo en la clase Array)

Generalmente, los métodos `sort()` y `sortOn()` modifican un conjunto. Si se desea ordenar un conjunto sin modificarlo, se debe pasar la constante `Array.RETURNINDEXEDARRAY` como parte del parámetro `options`. Esta opción ordena a los métodos que devuelvan un nuevo conjunto que refleje la ordenación y deje el conjunto original sin cambios. El conjunto devuelto por los métodos es un conjunto simple de números de índice que refleja el nuevo orden de clasificación y no contiene ningún elemento del conjunto original. Por ejemplo, para ordenar el conjunto `poets` por año de nacimiento sin modificar el conjunto, se debe incluir la constante `Array.RETURNINDEXEDARRAY` como parte del argumento pasado para el parámetro `options`.

El ejemplo siguiente almacena la información de índice devuelta en un conjunto denominado `indices` y utiliza el conjunto `indices` junto con el conjunto `poets` sin modificar para mostrar los poetas ordenados por año de nacimiento:

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
    var index:int = indices[i];
    trace(poets[index].name, poets[index].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Realización de consultas en un conjunto

Cuatro métodos de las clases Array y Vector (`concat()`, `join()`, `slice()` y `toString()`) consultan el conjunto para obtener información, pero no lo modifican. Los métodos `concat()` y `slice()` devuelven conjuntos nuevos, mientras que los métodos `join()` y `toString()` devuelven cadenas. El método `concat()` requiere una nuevo conjunto o lista de elementos como argumentos, y los combina con el conjunto existente para crear una nuevo conjunto. El método `slice()` tiene dos parámetros, denominados `startIndex` y `endIndex`, y devuelve un nuevo conjunto que contiene una copia de los elementos "extraídos" del conjunto existente. Se empieza por extraer el elemento con índice `startIndex` y el último elemento extraído es el que tiene el índice inmediatamente anterior a `endIndex`. Esto conlleva una repetición: el elemento con índice `endIndex` no se incluye en el valor devuelto.

En el ejemplo siguiente se utilizan `concat()` y `slice()` para crear conjuntos nuevos a partir de elementos de otros conjuntos:

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // output: alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // output: alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // output: alpha,beta,gamma
```

Se pueden utilizar los métodos `join()` y `toString()` para consultar el conjunto y devolver su contenido en forma de cadena. Si no se utiliza ningún parámetro para el método `join()`, los dos métodos se comportarán de forma idéntica: devolverán una cadena que contiene una lista delimitada por comas de todos los elementos del conjunto. El método `join()`, a diferencia del método `toString()`, admite un parámetro denominado `delimiter`, que permite elegir el símbolo que se debe utilizar como separador entre cada elemento de la cadena devuelta.

En el ejemplo siguiente se crea un conjunto denominado `rivers` y se llama a `join()` y `toString()` para devolver los valores del conjunto en forma de cadena. El método `toString()` se utiliza para devolver valores separados por comas (`riverCSV`), mientras que el método `join()` se utiliza para devolver valores separados por el carácter `+`.

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // output: Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // output: Nile+Amazon+Yangtze+Mississippi
```

Una característica del método `join()` que hay tener en cuenta es que las instancias anidadas de Array o Vector siempre se devuelven con los valores separados por comas, independientemente del separador que se especifique para los elementos del conjunto principal, como se indica en el siguiente ejemplo:

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // output: a+b,c,d+e
```

Conjuntos asociativos

Un conjunto asociativo, también denominado a veces *hash* o *asignación*, utiliza *claves* en lugar de un índice numérico para organizar los valores almacenados. Cada clave de un conjunto asociativo es una cadena única que se utiliza para acceder a un valor almacenado. Un conjunto asociativo es una instancia de la clase Object, lo que significa que cada clave corresponde a un nombre de propiedad. Los conjuntos asociativos son colecciones no ordenadas de pares formados por una clave y un valor. Hay que tener en cuenta en el código que las claves de un conjunto asociativo no tienen un orden específico.

ActionScript 3.0 también introduce un tipo avanzado de conjunto asociativo denominado *dictionary*. Los diccionarios, que son instancias de la clase Dictionary del paquete flash.utils, utilizan claves que pueden ser de cualquier tipo de datos. Es decir, las claves de diccionario no están limitadas a valores de tipo String.

Conjuntos asociativos con claves de tipo cadena

Existen dos formas de crear conjuntos asociativos en ActionScript 3.0. La primera es utilizar una instancia de Object. Al utilizar una instancia de Object, es posible inicializar el conjunto con un literal de objeto. Una instancia de la clase Object, conocida también como *objeto genérico*, es funcionalmente idéntica a un conjunto asociativo. El nombre de cada propiedad del objeto genérico actúa a modo de clave que permite acceder a un valor almacenado.

Este código crea un conjunto asociativo denominado `monitorInfo` y utiliza un literal de objeto para inicializar el conjunto con dos pares clave/valor.

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Si no es necesario inicializar el conjunto en el momento de declararlo, se puede utilizar el constructor Object para crear el conjunto de la manera siguiente:

```
var monitorInfo:Object = new Object();
```

Una vez creado un conjunto con un literal de objeto o el constructor de la clase Object, se pueden añadir valores nuevos al conjunto mediante el operador de acceso a conjunto (`[]`) o el operador punto (`.`). En el ejemplo siguiente se añaden dos valores nuevos a `monitorArray`:

```
monitorInfo["aspect ratio"] = "16:10"; // bad form, do not use spaces
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// output: 16:10 16.7 million
```

Hay que tener en cuenta que la clave denominada `aspect ratio` contiene un espacio en blanco. Esto es posible con el operador de acceso a conjunto (`[]`) pero genera un error si se intenta con el operador punto. No es recomendable utilizar espacios en los nombres de claves.

La segunda forma de crear un conjunto asociativo consiste en utilizar el constructor Array (o el constructor de cualquier clase dinámica) y posteriormente utilizar el operador de acceso a conjunto (`[]`) o el operador punto (`.`) para agregar pares de clave y valor al conjunto. Si se declara el conjunto asociativo con el tipo Array, no se podrá utilizar un literal de objeto para inicializar el conjunto. En el ejemplo siguiente se crea un conjunto asociativo denominado `monitorInfo` empleando el constructor de Array y se añaden claves denominadas `type` y `resolution`, junto con sus valores:

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

La utilización del constructor de Array para crear un conjunto asociativo no aporta ninguna ventaja. No se puede utilizar la propiedad `Array.length` ni ninguno de los métodos de la clase Array con los conjuntos asociativos, incluso si se utiliza el constructor de Array o el tipo de datos Array. Es mejor dejar el uso del constructor de Array para la creación de conjuntos indexados.

Conjuntos asociativos con claves de tipo objeto (Dictionaries)

Se puede utilizar la clase Dictionary para crear un conjunto asociativo que utilice objetos como claves en lugar de cadenas. Estos conjuntos se llaman a veces diccionarios, hashes o asignaciones. Por ejemplo, considérese una aplicación que determina la ubicación de un objeto Sprite a partir de su asociación con un contenedor específico. Se puede utilizar un objeto Dictionary para asignar cada objeto Sprite a un contenedor.

El código siguiente crea tres instancias de la clase Sprite que serán las claves del objeto Dictionary. A cada clave se le asigna un valor `GroupA` o `GroupB`. Los valores pueden ser de cualquier tipo de datos, pero en este ejemplo `GroupA` y`GroupB` son instancias de la clase Object. Posteriormente se podrá acceder al valor asociado con cada clave mediante el operador de acceso a conjunto (`[]`), como se indica en el código siguiente:

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// objects to use as keys
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// objects to use as values
var groupA:Object = new Object();
var groupB:Object = new Object();

// Create new key-value pairs in dictionary.
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
    trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
    trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
    trace("spr3 is in groupB");
}
```

Iteración con claves de tipo objeto

Puede repetir el contenido de un objeto Dictionary con un bucle `for..in` o un bucle `for each..in`. El bucle `for..in` permite la repetición en función de las claves, mientras que el bucle `for each..in` lo hace en función de los valores asociados con cada clave.

Utilice el bucle `for..in` para dirigir el acceso a las claves de tipo Object de un objeto Dictionary. También se puede acceder a los valores del objeto Dictionary con el operador de acceso a conjunto (`[]`). El código siguiente utiliza el ejemplo anterior del diccionario `groupMap` para mostrar la forma de repetición de un objeto Dictionary con el bucle `for..in`:

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
/* output:
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

Utilice el bucle `for each..in` para dirigir el acceso a los valores de un objeto Dictionary. El código siguiente también utiliza el diccionario `groupMap` para mostrar la forma de repetición de un objeto Dictionary con el bucle `for each..in`:

```
for each (var item:Object in groupMap)
{
    trace(item);
}
/* output:
[object Object]
[object Object]
[object Object]
*/
```

Claves de tipo objeto y administración de memoria

Adobe® Flash® Player y Adobe® AIR™ utilizan un sistema de eliminación de datos innecesarios para recuperar la memoria que ya no se está utilizando. Cuando ya no quedan referencias a un objeto, el objeto se convierte en disponible para la eliminación de datos innecesarios y se recupera la memoria la próxima vez que se ejecute el sistema de eliminación de datos innecesarios. Por ejemplo, el código siguiente crea un nuevo objeto y asigna una referencia al objeto a la variable `myObject`:

```
var myObject:Object = new Object();
```

Con que exista una referencia al objeto, el sistema de eliminación de datos innecesarios no recuperará la memoria ocupada por el objeto. Si se cambia el valor de `myObject` de forma que haga referencia a un objeto distinto o se establece en el valor `null`, la memoria ocupada por el objeto original se convierte en disponible para la eliminación de datos innecesarios, pero sólo si no hay otras referencias al objeto original.

Si se utiliza `myObject` como clave de un objeto Dictionary, se crea otra referencia al objeto original. Por ejemplo, el código siguiente crea dos referencias a un objeto: la variable `myObject` y la clave del objeto `myMap`:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

Para hacer que el objeto al que hace referencia `myObject` esté disponible para la eliminación de datos innecesarios, hay que eliminar todas las referencias a dicho objeto. En este caso, hay que cambiar el valor de `myObject` y eliminar la clave `myObject` de `myMap`, como se indica en el código siguiente:

```
myObject = null;
delete myMap[myObject];
```

Como alternativa, se puede utilizar el parámetro `useWeakReference` del constructor de Dictionary para convertir todas las claves del diccionario en *referencias débiles*. El sistema de eliminación de datos innecesarios no tiene en cuenta las referencias débiles, lo que significa que un objeto que sólo tenga referencias débiles estará disponible para la eliminación de datos innecesarios. Por ejemplo, en el código siguiente no es necesario eliminar la clave `myObject` de `myMap` para hacer que el objeto esté disponible para la eliminación de datos innecesarios:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // Make object eligible for garbage collection.
```

Conjuntos multidimensionales

Los conjuntos multidimensionales contienen otros conjuntos como elementos. Piense, por ejemplo, en una lista de tareas almacenadas en forma de conjunto de cadenas indexadas:

```
var tasks:Array = ["wash dishes", "take out trash"];
```

Si se desea almacenar una lista independiente de tareas por cada día de la semana, se puede crear un conjunto multidimensional con un elemento por cada día de la semana. Cada elemento contiene un conjunto indexado, similar al conjunto `tasks`, que almacena la lista de tareas. Se puede utilizar cualquier combinación de conjuntos indexados o asociativos en conjuntos multidimensionales. Los ejemplos de las secciones siguientes utilizan dos conjuntos indexados o un conjunto asociativo de conjuntos indexados. Se pueden probar las otras combinaciones como ejercicio.

Dos conjuntos indexados

Si se utilizan dos conjuntos indexados, se puede visualizar el resultado como una tabla o una hoja de cálculo. Los elementos del primer conjunto representan las filas de la tabla, mientras que los elementos del segundo conjunto representan las columnas.

Por ejemplo, el siguiente conjunto multidimensional utiliza dos conjuntos indexados para hacer un seguimiento de las listas de tareas para cada día de la semana. El primer conjunto, `masterTaskList`, se crea mediante el constructor de la clase Array. Cada elemento del conjunto representa un día de la semana, donde el índice 0 representa el lunes y el índice 6 representa el domingo. Estos elementos pueden considerarse como las filas de la tabla. Se puede crear la lista de tareas de cada día asignando un literal de conjunto a cada uno de los siete elementos creados en el conjunto `masterTaskList`. Los literales de conjunto representan las columnas en la tabla.

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

Puede acceder a elementos individuales de cualquiera de las listas de tareas mediante el operador de acceso a conjunto (`[]`). El primer conjunto de corchetes representa el día de la semana y el segundo conjunto de corchetes representa la lista de tareas para ese día. Por ejemplo, para recuperar la segunda tarea de la lista del miércoles, se debe utilizar primero el índice 2 correspondiente al miércoles y después el índice 1 correspondiente a la segunda tarea de la lista.

```
trace(masterTaskList[2][1]); // output: dentist
```

Para recuperar la primera tarea de la lista del domingo se debe utilizar el índice 6 correspondiente al domingo y el índice 0 correspondiente a la primera tarea de la lista.

```
trace(masterTaskList[6][0]); // output: mow lawn
```

Conjunto asociativo con un conjunto indexado

Para facilitar el acceso a los conjuntos individuales, se puede utilizar un conjunto asociativo para los días de la semana y un conjunto indexado para las listas de tareas. El uso de un conjunto asociativo permite utilizar la sintaxis con punto al hacer referencia a un día específico de la semana, pero a cambio hay un procesamiento adicional en tiempo de ejecución para acceder a cada elemento del conjunto asociativo. En el ejemplo siguiente se utiliza un conjunto asociativo como la base de una lista de tareas, con un par clave-valor para cada día de la semana:

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

La sintaxis con punto facilita la lectura del código al evitar la necesidad de utilizar varios juegos de corchetes.

```
trace(masterTaskList.Wednesday[1]); // output: dentist
trace(masterTaskList.Sunday[0]);// output: mow lawn
```

Puede repetir la lista de tareas utilizando un bucle `for..in`, pero debe utilizar el operador de acceso a conjunto (`[]`) en lugar de la sintaxis de puntos para acceder al valor asociado a cada clave. Como `masterTaskList` es un conjunto asociativo, los elementos no se recuperan necesariamente en el orden esperado, como se muestra en el siguiente ejemplo:

```
for (var day:String in masterTaskList)
{
    trace(day + ": " + masterTaskList[day])
}
/* output:
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

Clonación de conjuntos

La clase Array no tiene ningún método incorporado para hacer copias de conjuntos. Se puede crear una *copiasuperficial* de un conjunto llamando a los métodos `concat()` o `slice()` sin argumentos. En una copia superficial, si el conjunto original tiene elementos que son objetos, sólo se copian las referencias a los objetos, en lugar de los mismos objetos. La copia señala a los mismos objetos que el conjunto original. Los cambios realizados en los objetos se reflejan en ambos conjuntos.

En una *copia completa* también se copian los objetos del conjunto original, de forma que el nuevo conjunto no señale a los mismos objetos que el conjunto original. La copia completa requiere más de una línea de código, que normalmente ordenan la creación de una función. Dicha función se puede crear como una función de utilidad global o como un método de una subclase Array.

En el ejemplo siguiente se define una función denominada `clone()` que realiza una copia completa. Se utiliza un algoritmo de una técnica de programación común en Java. La función crea una copia completa serializando el conjunto en una instancia de la clase ByteArray y leyendo a continuación el conjunto en un nuevo conjunto. Esta función acepta un objeto de forma que se pueda utilizar tanto con conjuntos indexados como con conjuntos asociativos, como se indica en el código siguiente:

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
    var myBA:ByteArray = new ByteArray();
    myBA.writeObject(source);
    myBA.position = 0;
    return(myBA.readObject());
}
```

Temas avanzados

Ampliación de la clase Array

La clase Array es una de las pocas clases principales que no son finales, lo que significa que es posible crear una subclase de Array. Esta sección proporciona un ejemplo de cómo se puede crear una subclase de Array y se describen algunos de los problemas que pueden surgir durante el proceso.

Como se mencionó anteriormente, en ActionScript los conjuntos no tienen tipo, pero se puede crear una subclase de Array que acepte elementos de un solo tipo de datos específico. El ejemplo de las secciones siguientes define una subclase de Array denominada TypedArray que limita sus elementos a valores del tipo de datos especificado en el primer parámetro. La clase TypedArray se presenta simplemente como un ejemplo de cómo ampliar la clase Array y puede no ser adecuado para fines de producción por diversas razones. En primer lugar, la verificación de tipos se realiza en tiempo de ejecución, no en tiempo de compilación. En segundo lugar, cuando un método TypedArray encuentra un tipo no coincidente, se omite el tipo no coincidente y no se emite ninguna excepción, aunque los métodos pueden ser fácilmente modificados para emitir excepciones. Además, la clase no puede evitar el uso del operador de acceso a un conjunto para insertar valores de cualquier tipo en el conjunto. Por último, el estilo de programación favorece la simplicidad frente a la optimización del rendimiento.

Nota: *puede utilizar la técnica que se describe aquí para crear un conjunto de tipos. Sin embargo, se recomienda utilizar un objeto Vector. Una instancia de Vector es un conjunto de tipos y ofrece rendimiento y otras mejoras en comparación con la clase Array o cualquiera de sus subclases. La finalidad de esta argumentación es demostrar cómo se crea una subclase de Array.*

Declaración de la subclase

La palabra clave `extends` permite indicar que una clase es una subclase de Array. Una subclase de Array debe utilizar el atributo `dynamic`, igual que la clase Array. De lo contrario, la subclase no funcionará correctamente.

El código siguiente muestra la definición de la clase TypedArray, que contiene una constante en la que se almacena el tipo de datos, un método constructor y los cuatro métodos que pueden añadir elementos al conjunto. En este ejemplo se omite el código de cada método, pero se describe y explica completamente en las secciones siguientes:

```
public dynamic class TypedArray extends Array
{
    private const dataType:Class;

    public function TypedArray(...args) {}

    AS3 override function concat(...args):Array {}

    AS3 override function push(...args):uint {}

    AS3 override function splice(...args) {}

    AS3 override function unshift(...args):uint {}
}
```

Los cuatro métodos sustituidos utilizan el espacio de nombres AS3 en lugar del atributo `public`, ya que en este ejemplo se supone que la opción de compilador `-as3` está establecida en `true` y la opción de compilador `-es` está establecida en `false`. Esta es la configuración predeterminada para Adobe Flex Builder 3 y Adobe® Flash® CS4 Professional. Para obtener más información, consulte "El espacio de nombres AS3" en la página 125.

Los programadores expertos que prefieren utilizar herencia de prototipo pueden hacer dos pequeños cambios en la clase TypedArray para que se compile con la opción de compilador `-es` establecida en `true`. En primer lugar, deben quitarse todas las instancias del atributo `override` y debe sustituirse el espacio de nombres AS3 por el atributo `public`. En segundo lugar, debe sustituirse `Array.prototype` para las cuatro instancias de `super`.

Constructor de TypedArray

El constructor de la subclase supone un reto interesante, ya que debe aceptar una lista de argumentos de longitud arbitraria. El reto consiste en pasar los argumentos al superconstructor para crear el conjunto. Si se pasa la lista de argumentos en forma de conjunto, el superconstructor considerará que se trata de un solo argumento de tipo Array y el conjunto resultante siempre tendrá una longitud de 1 elemento. La manera tradicional de controlar las listas de argumentos es utilizar el método `Function.apply()`, que admite un conjunto de argumentos como segundo parámetro, pero la convierte en una lista de argumentos al ejecutar la función. Por desgracia, el método `Function.apply()` no se puede utilizar con constructores.

La única opción que queda es volver a generar la lógica del constructor de Array in el constructor de TypedArray. El código siguiente muestra el algoritmo utilizado en el constructor de clase Array, que se puede reutilizar en el constructor de la subclase de Array:

```
public dynamic class Array
{
    public function Array(...args)
    {
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen;
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")");
            }
            length = ulen;
        }
        else
        {
            length = n;
            for (var i:int=0; i < n; i++)
            {
                this[i] = args[i]
            }
        }
    }
}
```

El constructor de TypedArray comparte la mayor parte del código del constructor de Array, con tan sólo cuatro cambios. En primer lugar, la lista de parámetros incluye un nuevo parámetro requerido de tipo Class que permite especificar el tipo de datos del conjunto. En segundo lugar, el tipo de datos pasado al constructor se asigna a la variable `dataType`. En tercer lugar, en la sentencia `else`, el valor de la propiedad `length` se asigna después del bucle `for`, de forma que `length` incluya únicamente argumentos del tipo adecuado. Por último, el cuerpo del bucle `for` utiliza la versión sustituida del método `push()` de forma que sólo se añadan al conjunto los argumentos que tengan el tipo de datos correcto. En el siguiente ejemplo se muestra la función constructora de TypedArray:

```
public dynamic class TypedArray extends Array
{
    private var dataType:Class;
    public function TypedArray(typeParam:Class, ...args)
    {
        dataType = typeParam;
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")")
            }
            length = ulen;
        }
        else
        {
            for (var i:int=0; i < n; i++)
            {
                // type check done in push()
                this.push(args[i])
            }
            length = this.length;
        }
    }
}
```

Métodos sustituidos de TypedArray

La clase TypedArray reemplaza los cuatro métodos de la clase Array que pueden añadir elementos a un conjunto. En cada caso, el método sustituido añade una verificación de tipos que evita la adición de elementos que no tienen el tipo de datos correcto. Posteriormente, cada método llama a la versión de sí mismo de la superclase.

El método `push()` repite la lista de argumentos con un bucle `for..in` y realiza una verificación de tipos en cada argumento. Cualquier argumento que no sea del tipo correcto se quitará del conjunto `args` con el método `splice()`. Una vez finalizado el bucle `for..in`, el conjunto `args` sólo incluirá valores de tipo `dataType`. A continuación, se llama a la versión de `push()` de la superclase con el conjunto `args` actualizada, como se indica en el código siguiente:

```
AS3 override function push(...args):uint
{
    for (var i:* in args)
    {
        if (!(args[i] is dataType))
        {
            args.splice(i,1);
        }
    }
    return (super.push.apply(this, args));
}
```

El método `concat()` crea un objeto TypedArray temporal denominado `passArgs` para almacenar los argumentos que superen la verificación de tipos. Esto permite reutilizar el código de verificación de tipos que existe en el método `push()`. Un bucle `for..in` repite el conjunto `args` y llama a `push()` en cada argumento. Como `passArgs` es de tipo TypedArray, se ejecuta la versión de `push()` de TypedArray. A continuación, el método `concat()` llama a su propia versión de la superclase, como se indica en el código siguiente:

```
    AS3 override function concat(...args):Array
    {
        var passArgs:TypedArray = new TypedArray(dataType);
        for (var i:* in args)
        {
            // type check done in push()
            passArgs.push(args[i]);
        }
        return (super.concat.apply(this, passArgs));
    }
```

El método `splice()` admite una lista de argumentos arbitraria, pero los dos primeros argumentos siempre hacen referencia a un número de índice y al número de elementos que se desea eliminar. Por esta razón, el método `splice()` sustituido sólo hace la verificación de tipos para los elementos del conjunto `args` cuya posición de índice sea 2 o superior. Un aspecto interesante del código es que parece una llamada recursiva a `splice()` desde el bucle `for`, pero no es una llamada recursiva, ya que `args` es de tipo Array, no TypedArray, lo que significa que la llamada a `args.splice()` es una llamada a la versión del método de la superclase. Una vez que el bucle `for..in` haya finalizado, el conjunto `args` sólo incluirá valores del tipo correcto en posiciones cuyo índice sea 2 o superior, y `splice()` llamará a su propia versión de la superclase, como se indica en el siguiente código:

```
    AS3 override function splice(...args):*
    {
        if (args.length > 2)
        {
            for (var i:int=2; i< args.length; i++)
            {
                if (!(args[i] is dataType))
                {
                    args.splice(i,1);
                }
            }
        }
        return (super.splice.apply(this, args));
    }
```

El método `unshift()`, que añade elementos al principio de un conjunto, también acepta una lista de argumentos arbitraria. El método `unshift()` sustituido utiliza un algoritmo muy similar al utilizado por el método`push()`, como se indica en el siguiente ejemplo código:

```
    AS3 override function unshift(...args):uint
    {
        for (var i:* in args)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
        return (super.unshift.apply(this, args));
    }
}
```

Ejemplo: PlayList

El ejemplo PlayList ilustra técnicas para trabajar con conjuntos, en el contexto de una aplicación de lista de reproducción de música que administra una lista de canciones. Estas técnicas son:

- Creación de un conjunto indexado
- Añadir elementos a un conjunto indexado
- Ordenación de un conjunto de objetos por distintas propiedades y utilizando distintas opciones de ordenación
- Conversión de un conjunto en una cadena delimitada por caracteres

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación PlayList se encuentran en la carpeta Samples/PlayList. La aplicación consta de los siguientes archivos:

Archivo	Descripción
PlayList.mxml o PlayList.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/playlist/PlayList.as	Una clase que representa una lista de canciones. Utiliza un elemento Array para guardar la lista y gestiona la ordenación de los elementos de la lista.
com/example/programmingas3/playlist/Song.as	Un objeto de valor que representa información sobre una sola canción. Los elementos administrados por la clase PlayList son instancias de Song.
com/example/programmingas3/playlist/SortProperty.as	Una seudoenumeración cuyos valores disponibles representan las propiedades de la clase Song por las que una lista de objetos Song puede ordenarse.

Información general sobre la clase PlayList

La clase PlayList administra un conjunto de objetos Song. Dispone de métodos públicos con funcionalidad para añadir una canción a la lista de reproducción (el método `addSong()`) y ordenar las canciones de la lista (el método `sortList()`). Además, la clase incluye una propiedad de descriptor de acceso de sólo lectura, `songList`, que proporciona acceso al conjunto de canciones de la lista de reproducción. Internamente, la clase PlayList hace un seguimiento de sus canciones mediante una variable privada de tipo Array:

```
public class PlayList
{
    private var _songs:Array;
    private var _currentSort:SortProperty = null;
    private var _needToSort:Boolean = false;
    ...
}
```

Además de la variable `_songs` de tipo Array utilizada por la clase PlayList para hacer un seguimiento de su lista de canciones, otras dos variables privadas hacen un seguimiento de si la lista debe ser ordenada (`_needToSort`) y por qué propiedad está ordenada la lista de canciones en un momento dado (`_currentSort`).

Al igual que ocurre con los objetos, declarar una instancia de Array es sólo la mitad del trabajo de crear un objeto Array. Antes de acceder a las propiedades o métodos de una instancia de Array, hay que crear una instancia en el constructor de la clase PlayList.

```
public function PlayList()
{
    this._songs = new Array();
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
}
```

La primera línea del constructor crea una instancia de la variable `_songs` lista para usar. Además, se llama al método `sortList()` para establecer la propiedad por la que se va a ordenar inicialmente.

Añadir una canción a la lista

Cuando un usuario introduce una nueva canción en la aplicación, el código del formulario de entrada de datos llama al método `addSong()` de la clase PlayList.

```
/**
 * Adds a song to the playlist.
 */
public function addSong(song:Song):void
{
    this._songs.push(song);
    this._needToSort = true;
}
```

En `addSong()`, se llama al método `push()` del conjunto `_songs`, para añadir el objeto Song que se pasó a `addSong()` como un elemento nuevo del conjunto. Con el método `push()` se añade el nuevo elemento al final del conjunto, independientemente de la ordenación que se haya aplicado previamente. Esto significa que tras llamar al método `push()`, es probable que la lista de canciones ya no esté ordenada correctamente, por lo que se establece la variable `_needToSort` en `true`. En teoría, el método `sortList()` podría llamarse inmediatamente, lo que eliminaría la necesidad de controlar si la lista está ordenada o no en un momento determinado. No obstante, en la práctica no es necesario ordenar la lista de canciones hasta inmediatamente antes de recuperarla. Al aplazar la operación de ordenación, la aplicación no realiza una ordenación innecesaria si, por ejemplo, se añaden varias canciones a la lista antes de recuperarla.

Ordenación de la lista de canciones

Como las instancias de Song administradas por la lista de reproducción son objetos complejos, es posible que los usuarios de la aplicación deseen ordenar la lista de reproducción según distintas propiedades, como el título de la canción o el año de publicación. En la aplicación PlayList, la tarea de ordenación de la lista de canciones tiene tres partes: identificación de la propiedad con la que se debe ordenar la lista, indicación de las opciones de ordenación que se deben utilizar al ordenar con dicha propiedad y la ejecución de la operación real de ordenación.

Propiedades para la ordenación

Un objeto Song hace un seguimiento de varias propiedades, como el título de la canción, el artista, el año de publicación, el nombre de archivo y un conjunto de géneros a los que pertenece la canción seleccionada por el usuario. De éstas, sólo las tres primeras son prácticas para ordenar. Para mayor comodidad de los desarrolladores, el ejemplo incluye la clase SortProperty, que actúa como una enumeración con valores que representan las propiedades disponibles para ordenar.

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

La clase SortProperty contiene tres constantes, `TITLE`, `ARTIST` y `YEAR`, cada una de las cuales almacena una cadena que contiene el nombre real de propiedad de la clase Song asociada que se puede utilizar para ordenar. En el resto del código, siempre que se indique una propiedad para ordenar, se hará con el miembro de la enumeración. Por ejemplo, en el constructor de PlayList, la lista se ordena inicialmente llamando al método `sortList()` de la manera siguiente:

```
// Set the initial sorting.
this.sortList(SortProperty.TITLE);
```

Como la propiedad para ordenar se especifica como `SortProperty.TITLE`, las canciones se ordenan por su título.

Ordenación por propiedades y especificación de opciones de ordenación

El trabajo de ordenar la lista de canciones lo realiza la clase PlayList en el método `sortList()`, de la manera siguiente:

```
/**
 * Sorts the list of songs according to the specified property.
 */
public function sortList(sortProperty:SortProperty):void
{
    ...
    var sortOptions:uint;
    switch (sortProperty)
    {
        case SortProperty.TITLE:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.ARTIST:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.YEAR:
            sortOptions = Array.NUMERIC;
            break;
    }

    // Perform the actual sorting of the data.
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // Save the current sort property.
    this._currentSort = sortProperty;

    // Record that the list is sorted.
    this._needToSort = false;
}
```

Al ordenar por título o por artista, tiene sentido ordenar alfabéticamente, pero al ordenar por año, es más lógico realizar una ordenación numérica. La sentencia `switch` se utiliza para definir la opción de ordenación apropiada, almacenada en la variable `sortOptions`, según el valor especificado en el parámetro `sortProperty`. En este caso también se utilizan los miembros de la enumeración designados para distinguir entre propiedades, en lugar de utilizar valores especificados en el código.

Con la propiedad de ordenación y las opciones de ordenación determinadas, el conjunto `_songs` se ordena llamando a su método `sortOn()` y pasándole esos dos valores como parámetros. Se registra la propiedad de ordenación actual, ya que la lista de canciones está ordenada actualmente.

Combinación de elementos de conjunto en una cadena delimitada por caracteres

Además de utilizar un conjunto para mantener la lista de canciones de la clase PlayList, en este ejemplo también se utilizan conjuntos en la clase Song para administrar la lista de géneros a los que pertenece una canción determinada. Considérese este fragmento de la definición de la clase Song:

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String, genres:Array)
{
    ...
    // Genres are passed in as an array
    // but stored as a semicolon-separated string.
    this._genres = genres.join(";");
}
```

Al crear una nueva instancia de Song, se define el parámetro `genres` que se utiliza para especificar el género (o los géneros) al que pertenece la canción como una instancia de Array. Esto hace que sea cómodo agrupar varios géneros en una sola variable que se puede pasar al constructor. No obstante, internamente la clase Song mantiene los géneros en la variable privada `_genres` como una instancia de tipo String de valores separados por signos de punto y coma. El parámetro Array se convierte en una cadena de valores separados por signos de punto y coma llamando a su método `join()` con el valor de literal de cadena `";"` como delimitador especificado.

Con este mismo símbolo, los descriptores de acceso de `genres` permiten establecer o recuperar géneros como un conjunto:

```
    public function get genres():Array
    {
        // Genres are stored as a semicolon-separated String,
        // so they need to be transformed into an Array to pass them back out.
        return this._genres.split(";");
    }
    public function set genres(value:Array):void
    {
        // Genres are passed in as an array,
        // but stored as a semicolon-separated string.
        this._genres = value.join(";");
    }
```

Los descriptores de acceso `set` de `genres` se comportan exactamente igual que el constructor; aceptan un conjunto y llaman al método `join()` para convertirlo en una cadena de valores separados por signos de punto y coma. El descriptor de acceso `get` realiza la operación contraria: se llama el método `split()` de la variable `_genres` y la cadena se divide en un conjunto de valores utilizando el delimitador especificado (valor de cadena literal `";"`, como antes).

Capítulo 9: Gestión de errores

Gestionar errores implica crear una lógica en la aplicación que responda a errores o los corrija. Dichos errores se generan cuando se compila una aplicación o se ejecuta una aplicación compilada. Si la aplicación gestiona errores, se produce *algo* como respuesta a la detección de un error, en lugar de dejar que el proceso genere el error sin emitir ninguna respuesta. Si se usa correctamente, la gestión de errores ayuda a proteger la aplicación y sus usuarios de comportamientos inesperados.

Sin embargo, la gestión de errores es una categoría extensa que incluye respuestas a muchas clases de errores generados durante la compilación o en tiempo de ejecución. En este capítulo se analiza cómo gestionar errores en tiempo de ejecución, los distintos tipos de errores que se pueden generar y las ventajas del nuevo sistema de gestión de errores en ActionScript 3.0. En el capítulo también se explica cómo puede implementar sus propias estrategias personalizadas de gestión de errores para sus aplicaciones.

Fundamentos de la gestión de errores

Introducción a la gestión de errores

Un error en tiempo de ejecución es algo que no funciona en el código de ActionScript y que impide la ejecución del contenido en Adobe® Flash® Player o Adobe® AIR™. Para asegurarse de que los usuarios pueden ejecutar correctamente el código ActionScript, se debe añadir a la aplicación código para gestionar errores, es decir, para corregirlos o al menos indicar al usuario qué ha sucedido. Este proceso se denomina *gestión de errores*.

La gestión de errores es un campo amplio que incluye respuestas a muchas clases de errores generados durante la compilación o en tiempo de ejecución. Los errores que se producen en tiempo de compilación suelen ser más fáciles de identificar; se deben corregir para completar el proceso de creación de un archivo SWF. En este capítulo no se estudian los errores en tiempo de compilación; para obtener más información sobre la escritura de código que no incluye errores en tiempo de compilación, consulte "El lenguaje ActionScript y su sintaxis" en la página 38 y "Programación orientada a objetos con ActionScript" en la página 93. Este capítulo se centra en los errores en tiempo de ejecución.

La detección de errores en tiempo de ejecución puede resultar más difícil, ya que para que se produzcan, el código debe ejecutarse. Si un segmento de su programa tiene varias ramas de código, como una sentencia `if..thenthen..else`, debe comprobar todas las condiciones posibles, con todos los valores de entrada posibles que puedan utilizar los usuarios reales, con el fin de confirmar que el código no contenga errores.

Los errores en tiempo de ejecución se pueden dividir en dos categorías: los *errores de programa* son errores del código ActionScript, como la especificación del tipo de datos incorrecto para un parámetro de método; los *errores lógicos* son errores de la lógica (la comprobación de datos y la manipulación de valores) del programa, como la utilización de la fórmula incorrecta para calcular tipos de interés en una aplicación bancaria. Los dos tipos de error pueden detectarse y corregirse si se prueba la aplicación minuciosamente.

Lo ideal sería que se identificaran y se eliminaran todos los errores de la aplicación antes de que pasara a disposición de los usuarios finales. Sin embargo, no todos los errores pueden predecirse o evitarse. Por ejemplo, imagine que la aplicación ActionScript carga información de un sitio Web determinado que se encuentra fuera de su control. Si en algún momento dicho sitio no está disponible, la parte de la aplicación que depende de los datos externos no se comportará correctamente. El aspecto más importante de la gestión de errores implica prepararse para estos casos desconocidos, así como gestionarlos adecuadamente para que los usuarios puedan seguir utilizando la aplicación, o al menos obtengan un mensaje de error claro que indique por qué no funciona.

Los errores en tiempo de ejecución se representan de dos formas en ActionScript:

- Clases Error: muchos errores tienen una clase de error asociada a ellos. Cuando se produce un error, Flash Player o AIR crean una instancia de la clase de error específica asociada a dicho error. El código puede utilizar información contenida en ese objeto de error para generar una respuesta adecuada al error.
- Eventos de error: a veces se produce un error cuando Flash Player o Adobe AIR activarían normalmente un evento. En estos casos, Flash Player y Adobe AIR activan un evento de error. Al igual que en otros eventos, cada evento de error tiene una clase asociada y Flash Player y Adobe AIR transmiten una instancia de dicha clase a los métodos que están suscritos al evento de error.

Para determinar si un método concreto puede desencadenar un error o un evento de error, consulte la entrada del método en la Referencia del lenguaje y componentes ActionScript 3.0.

Tareas comunes de la gestión de errores

A continuación se describen tareas comunes relativas a errores que posiblemente se deban realizar con el código:

- Escribir código para gestionar errores
- Probar errores, detectarlos y volverlos a emitir
- Definir una clase de error personalizada
- Responder al error y a los eventos de error

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Asincrónico: comando de programa, como una llamada de método que no proporciona un resultado inmediato; en su lugar, origina un resultado (o error) en forma de evento.
- Capturar: cuando se produce una excepción (error en tiempo de ejecución) y el código la reconoce, se dice que el código *captura* la excepción. Una vez capturada la excepción, Flash Player y Adobe AIR dejan de notificar la excepción a otro código ActionScript.
- Versión de depuración: versión especial de Flash Player o Adobe AIR (ADL) que incluye código para notificar a los usuarios errores en tiempo de ejecución. En la versión estándar de Flash Player o Adobe AIR (la que tienen la mayoría de los usuarios), se omiten los errores no gestionados por el código ActionScript. En las versiones de depuración (que se incluye en Adobe Flash CS4 Professional y Adobe Flex), aparece un mensaje de advertencia cuando se produce un error no gestionado.
- Excepción: error que se produce cuando se ejecuta un programa y que el entorno en tiempo de ejecución (es decir, Flash Player o Adobe AIR) no puede resolver por sí mismo.
- Nueva emisión: cuando el código detecta una excepción, Flash Player y Adobe AIR no vuelven a notificar otros objetos de la excepción. Si resulta importante notificar la excepción a otros objetos, el código debe *volver a emitir* la excepción para que se vuelva a iniciar el proceso de notificación.

- Sincrónico: comando de programa, como una llamada de método, que proporciona un resultado inmediato (o emite inmediatamente un error), lo que implica que la respuesta puede utilizarse con el mismo bloque de código.
- Emitir: la operación de notificar a Flash Player o Adobe AIR (y, por lo tanto, notificar a otros objetos y al código ActionScript) que se ha producido un error, se conoce como *emitir* un error.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Prácticamente todos los listados de código de este capítulo incluyen la llamada a la función `trace()` apropiada. Para probar los listados de código de este capítulo:

1. Cree un documento de Flash vacío.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Ejecute el programa seleccionando Control > Probar película.

   El resultado de las funciones `trace()` del código se ve en el panel Salida.

Algunos de los últimos ejemplos de código son más complejos y se han programado como una clase. Para probar estos ejemplos:

1. Cree un documento de Flash vacío y guárdelo en el equipo.
2. Cree un nuevo archivo de ActionScript y guárdelo en el mismo directorio que el documento de Flash. El nombre del archivo debe coincidir con el nombre de la clase del listado de código. Por ejemplo, si el listado de código define una clase denominada ErrorTest, use el nombre ErrorTest.as para guardar el archivo de ActionScript.
3. Copie el listado de código en el archivo de ActionScript y guarde el archivo.
4. En el documento de Flash, haga clic en una parte vacía del escenario o espacio de trabajo para activar el inspector de propiedades del documento.
5. En el inspector de propiedades, en el campo Clase de documento, escriba el nombre de la clase de ActionScript que copió del texto.
6. Ejecute el programa seleccionando Control > Probar película.

   Verá el resultado del ejemplo en el panel Salida (si se utiliza la función `trace()` en el código del ejemplo) o en un campo de texto creado por el código del ejemplo.

Estas técnicas para probar los listados de código de ejemplo se describen de forma detallada en "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

Tipos de errores

Al desarrollar y ejecutar aplicaciones, se detectan diferentes tipos de errores, con su correspondiente terminología. En la tabla siguiente se presentan los principales tipos de errores y sus términos relacionados:

- El compilador de ActionScript captura los *errores en tiempo de compilación* durante la compilación del código. Estos errores se producen cuando los problemas sintácticos del código impiden crear la aplicación.

- Los *errores en tiempo de ejecución* se producen al ejecutar la aplicación tras compilarla. Los errores en tiempo de ejecución representan errores causados mientras un archivo SWF se reproduce en Adobe Flash Player o Adobe AIR. En la mayoría de los casos, es posible gestionar los errores en tiempo de ejecución conforme suceden, informando sobre ellos al usuario y adoptando medidas para que la aplicación se siga ejecutando. Si el error es grave (por ejemplo, no se puede establecer una conexión con un sitio Web remoto o cargar los datos requeridos), se puede recurrir a la gestión de errores para que la aplicación finalice sin problemas.
- Los *errores sincrónicos* son errores en tiempo de ejecución que se producen cuando se invoca una función. Por ejemplo, si se intenta utilizar un método específico y el método que se transmite al argumento no es válido, Flash Player genera una excepción. La mayor parte de los errores se producen de forma sincrónica (cuando se ejecuta la sentencia) y el flujo del control se pasa inmediatamente a la sentencia `catch` más apropiada.

  Por ejemplo, el fragmento de código siguiente genera un error en tiempo de ejecución porque no se llama al método `browse()` antes de que el programa intente cargar un archivo:

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
      fileRef.upload("http://www.yourdomain.com/fileupload.cfm");
  }
  catch (error:IllegalOperationError)
  {
      trace(error);
      // Error #2037: Functions called in incorrect sequence, or earlier
      // call was unsuccessful.
  }
  ```

  En este caso, un error en tiempo de ejecución se genera sincrónicamente, ya que Flash Player determinó que el método `browse()` no se llamó antes de intentar cargar el archivo.

  Para obtener información detallada sobre la gestión de errores sincrónicos, consulte “Gestión de errores sincrónicos en una aplicación” en la página 193.

- Los *erroresasincrónicos* son errores en tiempo de ejecución, que se producen en varios puntos de la ejecución de la aplicación; generan eventos que son capturados por los detectores de eventos. En las operaciones asincrónicas, una función inicia una operación, pero no espera a que se complete. Se puede crear un detector de eventos de error para esperar a que la aplicación o el usuario realicen alguna operación; si ésta falla, el error se captura con un detector de eventos y se responde al evento de error. A continuación, el detector de eventos llama a una función de controlador de eventos para responder al evento de error de una manera útil. Por ejemplo, el controlador de eventos puede iniciar un cuadro de diálogo que solicite al usuario que resuelva el error.

  Considérese el ejemplo de error sincrónico relativo a la carga de archivo presentado anteriormente. Si se llama correctamente al método `browse()` antes de iniciar una carga de archivo, Flash Player distribuirá varios eventos. Por ejemplo, cuando se inicia una carga, se distribuye el evento `open`. Si la operación de carga de archivo finaliza correctamente, se distribuye el evento `complete`. Puesto que la gestión de eventos es asincrónica (es decir, que no se produce en momentos específicos, conocidos y designados previamente), hay que utilizar el método `addEventListener()` para detectar estos eventos específicos, tal como se muestra en el código siguiente:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
    trace("...select...");
    var request:URLRequest = new URLRequest("http://www.yourdomain.com/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    event.target.upload(request.url);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

Para obtener información detallada sobre la gestión de errores asincrónicos, consulte "Respuesta al estado y a los eventos de error" en la página 198.

- Las *excepciones no capturadas* son errores generados sin una lógica (como una sentencia `catch`) que pueda responder a ellos. Si la aplicación genera un error y no se encuentran controladores de eventos o sentencias `catch` en el nivel actual o superior para gestionar el error, éste se considera una excepción no capturada.

  De forma predeterminada, Flash Player omite en tiempo de ejecución los errores no capturados e intenta seguir reproduciéndose si el error no detiene el archivo SWF actual, ya que los usuarios no tienen por qué ser capaces de resolver el error necesariamente. El proceso de omitir un error no capturado se denomina "generar errores sin mensaje" y puede complicar la depuración de aplicaciones. La versión de depuración de Flash Player responde a un error sin capturar mediante la finalización del script actual y la visualización de dicho error en la salida de la sentencia `trace`, o bien mediante la especificación del mensaje de error en un archivo de registro. Si el objeto de excepción es una instancia de la clase Error o una de sus subclases, se invocará el método `getStackTrace()` y se mostrará la información de seguimiento de la pila en la salida de la sentencia trace o en un archivo de registro. Para obtener más información sobre el uso de la versión de depuración de Flash Player, consulte "Trabajo con las versiones de depuración de Flash Player y AIR" en la página 192.

Gestión de errores en ActionScript 3.0

Puesto que muchas aplicaciones pueden ejecutarse sin crear la lógica para gestionar errores, los desarrolladores pueden sentirse tentados de aplazar la aplicación de la gestión de errores en sus aplicaciones. Sin embargo, sin la gestión de errores, una aplicación puede bloquearse fácilmente o frustrar al usuario si algo no funciona de la forma prevista. ActionScript 2.0 presenta una clase Error que permite crear una lógica en funciones personalizadas y generar una excepción con un mensaje específico. Puesto que la gestión de errores es esencial para facilitar la utilización de las aplicaciones, ActionScript 3.0 incluye una arquitectura ampliada para capturar errores.

Nota: *aunque en la Referencia del lenguaje y componentes ActionScript 3.0 se documentan las excepciones que generan muchos métodos, puede que no se incluyan todas las excepciones posibles de cada método. Puede que un método genere una excepción para errores de sintaxis u otros problemas que no se menciona explícitamente en la descripción del método, aun cuando en ésta se enumeren algunas de las excepciones.*

Elementos de la gestión de errores de ActionScript 3.0

ActionScript 3.0 incluye muchas herramientas para la gestión de errores:

- Clases Error. ActionScript 3.0 incluye una amplia gama de clases Error que amplían el ámbito de situaciones que pueden producir objetos de error. Cada clase Error ayuda a las aplicaciones a gestionar y responder a condiciones de error específicas, ya sean relativas a errores de sistema (como una condición MemoryError), errores de codificación (como una condición ArgumentError) errores de red y comunicación (como una condición URIError), o bien otras situaciones. Para obtener más información sobre cada clase, consulte "Comparación de las clases Error" en la página 201.
- Menos errores sin mensaje. En versiones anteriores de Flash Player, los errores se generaban y mostraban sólo si se usaba explícitamente la sentencia `throw`. En Flash Player 9 (y versiones posteriores) y en Adobe AIR, los métodos y las propiedades de ActionScript generan errores en tiempo de ejecución que permitirán gestionar estas excepciones de forma más eficaz cuando se produzcan, así como reaccionar individualmente ante cada excepción.
- Se muestran mensajes de error claros durante la depuración. Si se usa la versión de depuración de Flash Player o Adobe AIR, el código o las situaciones que produzcan problemas generarán mensajes de error detallados que permitirán identificar fácilmente los motivos del error en un bloque de código determinado. Esto permite optimizar la corrección de errores. Para obtener más información, consulte "Trabajo con las versiones de depuración de Flash Player y AIR" en la página 192.
- Los errores precisos permiten que los usuarios vean mensajes de error claros en tiempo de ejecución. En las versiones anteriores de Flash Player, el método `FileReference.upload()` devolvía un valor booleano `false` si la llamada `upload()` era incorrecta, e indicaba uno de cinco errores posibles. Si se produce un error al llamar al método `upload()` en ActionScript 3.0, se puede generar uno de cuatro errores específicos, lo que permite mostrar mensajes de error más precisos a los usuarios finales.
- Gestión de errores mejorada. Se generan errores diferentes para una gran variedad de situaciones habituales. Por ejemplo, en ActionScript 2.0, antes de que se llenara un objeto FileReference, la propiedad `name` tenía el valor `null`; de este modo, antes de poder usar o mostrar la propiedad `name`, es necesario asegurarse de que el valor está establecido y no es `null`. En ActionScript 3.0, si se intenta acceder a la propiedad `name` antes de que se especifique su valor, Flash Player generar un error de tipo IllegalOperationError, que indica que no se ha establecido el valor y que se pueden usar bloques `try.. catch..finally` para gestionar el error. Para obtener más información, consulte "Utilización de sentencias try..catch..finally" en la página 193.
- No se producen problemas de rendimiento significativos. La utilización de los bloques `try..catch..finally` para gestionar errores no consume recursos adicionales (o consume muy pocos recursos) en comparación con las versiones anteriores de ActionScript.
- Existe una clase ErrorEvent que permite crear detectores de eventos de errores asincrónicos específicos. Para obtener más información, consulte "Respuesta al estado y a los eventos de error" en la página 198.

Estrategias de gestión de errores

Mientras la aplicación no detecte una situación problemática, se podrá ejecutar correctamente si no se crea una lógica de gestión de errores en el código. Sin embargo, si no se gestionan los errores activamente y la aplicación detecta un problema, los usuarios nunca sabrán por qué falla la aplicación.

Hay diferentes formas de abordar la gestión de errores en la aplicación. En la lista siguiente se resumen las tres opciones principales para gestionar errores:

- Uso de las sentencias `try..catch..finally`. Estas sentencias capturarán los errores cuando se produzcan. Se pueden anidar las sentencias en una jerarquía para capturar excepciones en varios niveles de la ejecución del código. Para obtener más información, consulte "Utilización de sentencias try..catch..finally" en la página 193

- Creación de objetos de error personalizados. Se puede usar la clase Error para crear objetos de error personalizados, lo que permite hacer un seguimiento de operaciones específicas de la aplicación que no estén incluidas en los tipos de error integrados. Posteriormente puede utilizar sentencias `try..catch..finally` en sus propios objetos de error personalizados. Para obtener más información, consulte "Creación de clases de error personalizadas" en la página 197.
- Especificación de detectores y controladores de eventos para responder a eventos de error. Mediante esta estrategia, se pueden crear controladores de error globales que permiten gestionar eventos similares sin duplicar demasiado código en bloques `try..catch..finally`. Además, es más probable capturar errores asincrónicos con este método. Para obtener más información, consulte "Respuesta al estado y a los eventos de error" en la página 198.

Trabajo con las versiones de depuración de Flash Player y AIR

Adobe ofrece a los desarrolladores ediciones especiales de Flash Player y Adobe AIR para ayudar en las operaciones de depuración. Al instalar Adobe Flash CS4 Professional o Adobe Flex Builder 3, se obtiene una copia de la versión de depuración de Flash Player. También se obtiene la versión de depuración de Adobe AIR, denominada ADL, al instalar una de estas herramientas o como parte del SDK de Adobe AIR.

Existe una diferencia notable en la forma en que las versiones de depuración y la versiones comerciales de Flash Player y Adobe AIR indican los errores. Las versiones de depuración muestran el tipo de error (como Error, IOError, o EOFError de carácter genérico), el número de error y un mensaje de error legible para el usuario. Las versiones comerciales muestran únicamente el tipo y número de error. Por ejemplo, considérese el fragmento de código siguiente:

```
try
{
    tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
    tf.text = error.toString();
}
```

Si el método `readBoolean()` genera un error de tipo EOFError en la versión de depuración de Flash Player, se mostrará el mensaje siguiente en el campo de texto `tf`: "EOFError: Error #2030: Se ha detectado el final del archivo".

El mismo código en una versión comercial de Flash Player o Adobe AIR mostraría el siguiente texto: "EOFError: Error #2030."

Para minimizar los recursos y el tamaño en las versiones comerciales, no se muestran cadenas de mensajes de error. El número de error se puede consultar en la documentación (apéndices de la Referencia del lenguaje y componentes ActionScript 3.0) para conocer el significado del mensaje de error. También es posible reproducir el error con las versiones de depuración de Flash Player y AIR para ver el mensaje completo.

Gestión de errores sincrónicos en una aplicación

La gestión de errores más habitual hace referencia a la lógica de gestión de errores sincrónicos, en la que se insertan sentencias en el código para capturar errores sincrónicos en tiempo de ejecución. El tipo de gestión de errores permite que la aplicación observe los errores en tiempo de ejecución y se recupere cuando fallen las funciones. La lógica para capturar un error sincrónico incluye sentencias `try..catch..finally`, que literalmente intentan una operación, capturan cualquier respuesta de error de Flash Player o Adobe AIR y finalmente ejecutan alguna otra operación para administrar la operación fallida.

Utilización de sentencias try..catch..finally

Cuando trabaje con errores en tiempo de ejecución sincrónicos, utilice sentencias `try..catch..finally` para capturar errores. Si se produce un error de tiempo de ejecución, Flash Player o Adobe AIR generan una excepción, lo que significa que se suspende la ejecución normal y se crea un objeto especial de tipo Error. El objeto Error se genera en el primer bloque `catch` disponible.

La sentencia `try` incluye sentencias que podrían provocar errores. La sentencia `catch` siempre se debe utilizar con una sentencia `try`. Si se detecta un error en una de las sentencias del bloque `try`, se ejecutarán las sentencias `catch` asociadas a la sentencia `try`.

La sentencia `finally` incluye sentencias que se ejecutarán independientemente de que se produzcan errores en el bloque `try`. Si no hay errores, las sentencias del bloque `finally` se ejecutarán una vez que hayan finalizado las sentencias del bloque `try`. Si se producen errores, primero se ejecutará la sentencia `catch` correspondiente, seguida de las instancias del bloque `finally`.

El código siguiente muestra la sintaxis necesaria para usar las sentencias `try..catch..finally`:

```
try
{
    // some code that could throw an error
}
catch (err:Error)
{
    // code to react to the error
}
finally
{
    // Code that runs whether or not an error was thrown. This code can clean
    // up after the error, or take steps to keep the application running.
}
```

Cada sentencia `catch` identifica el tipo de excepción específico que gestiona. La sentencia `catch` puede especificar sólo clases de errores que son subclases de la clase Error. Cada sentencia `catch` se comprueba por orden. Sólo se ejecutará la primera sentencia `catch` que coincida con el tipo de error generado. Dicho de otra forma, si primero se comprueba la clase Error de nivel superior y luego la subclase de la clase Error, únicamente coincidirá la clase Error de nivel superior. En el código siguiente se muestra este escenario:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
```

El código anterior muestra el resultado siguiente:

```
<Error> I am an ArgumentError
```

Para capturar correctamente la clase de error ArgumentError, es necesario asegurarse de que los tipos de error más específicos se enumeran primero, y que los más genéricos aparecen después, tal como se muestra en el código siguiente:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
```

Varios métodos y propiedades de la API de Flash Player generan errores en tiempo de ejecución si detectan errores durante la ejecución. Por ejemplo, el método `close()` de la clase Sound genera un error IOError si no puede cerrar el flujo de audio, tal como se muestra en el código siguiente:

```
var mySound:Sound = new Sound();
try
{
    mySound.close();
}
catch (error:IOError)
{
    // Error #2029: This URLStream object does not have an open stream.
}
```

A medida que el usuario se familiarice con la Referencia del lenguaje y componentes ActionScript 3.0, reconocerá los métodos que generan excepciones, tal y como se indica en la descripción de cada método.

La sentencia throw

Flash Player y Adobe AIR generan excepciones cuando localizan errores en la aplicación en tiempo de ejecución. Asimismo, se pueden generar excepciones explícitamente mediante la sentencia `throw`. Si se generan errores de forma explícita, Adobe recomienda generar instancias de la clase Error o sus subclases. En el código siguiente se muestra una sentencia `throw` que genera una instancia de la clase Error, `MyErr`, y que termina por llamar a la función `myFunction()` como respuesta tras la generación del error:

```
var MyError:Error = new Error("Encountered an error with the numUsers value", 99);
var numUsers:uint = 0;
try
{
    if (numUsers == 0)
    {
        trace("numUsers equals 0");
    }
}
catch (error:uint)
{
    throw MyError; // Catch unsigned integer errors.
}
catch (error:int)
{
    throw MyError; // Catch integer errors.
}
catch (error:Number)
{
    throw MyError; // Catch number errors.
}
catch (error:*)
{
    throw MyError; // Catch any other error.
}
finally
{
    myFunction(); // Perform any necessary cleanup here.
}
```

Debe tenerse en cuenta que las sentencias `catch` se ordenan de manera que los tipos de datos más específicos se muestren en primer lugar. Si la sentencia `catch` del tipo de datos Number se muestra primero, las sentencias `catch` de los tipos de datos uint e int nunca se ejecutarán.

Nota: *en el lenguaje de programación Java, cada función que puede generar una excepción debe declararlo y enumerar las clases de excepción generables en una cláusula* `throws` *asociada a la declaración de la función. ActionScript no requiere la declaración de las excepciones que puede generar una función.*

Visualización de un mensaje de error simple

Una de las mayores ventajas del nuevo modelo de eventos de error y excepción radica en que permite indicar a los usuarios cuándo y por qué falla una acción. La parte del usuario consiste en escribir el código para que se muestre el mensaje y se ofrezcan opciones a modo de respuesta.

El siguiente código incluye una sencilla sentencia `try..catch` para mostrar el error en un campo de texto:

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;

    public class SimpleError extends Sprite
    {
        public var employee:XML =
            <EmpCode>
                <costCenter>1234</costCenter>
                <costCenter>1-234</costCenter>
            </EmpCode>;

        public function SimpleError()
        {
            try
            {
                if (employee.costCenter.length() != 1)
                {
                    throw new Error("Error, employee must have exactly one cost center assigned.");
                }
            }
            catch (error:Error)
            {
                var errorMessage:TextField = new TextField();
                errorMessage.autoSize = TextFieldAutoSize.LEFT;
                errorMessage.textColor = 0xFF0000;
                errorMessage.text = error.message;
                addChild(errorMessage);
            }
        }
    }
}
```

Al usar una mayor gama de clases de error y errores de compilador incorporados, ActionScript 3.0 proporciona más información que las versiones anteriores acerca del motivo por el que falla algo. Esto permite crear aplicaciones más estables con una mejor gestión de errores.

Regeneración de errores

Al crear aplicaciones, pueden darse varias circunstancias en las que sea necesario volver a generar un error, si éste no consigue gestionarse adecuadamente. Por ejemplo, el siguiente código muestra un bloque anidado `try..catch`, que regenera un error de tipo ApplicationError personalizado si el bloque anidado `catch` no puede gestionar el error:

```
try
{
    try
    {
        trace("<< try >>");
        throw new ArgumentError("some error which will be rethrown");
    }
    catch (error:ApplicationError)
    {
        trace("<< catch >> " + error);
        trace("<< throw >>");
        throw error;
    }
    catch (error:Error)
    {
        trace("<< Error >> " + error);
    }
}
catch (error:ApplicationError)
{
    trace("<< catch >> " + error);
}
```

La salida del fragmento anterior será de esta forma:

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

El bloque `try` anidado genera un error ApplicationError personalizado que captura el bloque `catch` posterior. Este bloque `catch` anidado puede intentar gestionar el error y, si no lo consigue, puede generar el objeto ApplicationError en el bloque `try..catch` en el que está contenido.

Creación de clases de error personalizadas

Se puede ampliar una de las clases Error estándar para crear clases de error especializadas propias en ActionScript. Existen varias razones por las que se crean clases de error personalizadas:

- Para identificar errores o grupos de errores específicos que son exclusivos de la aplicación.

  Por ejemplo, es posible que se deseen realizar diferentes acciones en el caso de errores generados por el código personalizado, además de los que captura Flash Player o Adobe AIR. Se puede crear una subclase de la clase Error para hacer un seguimiento del nuevo tipo de datos de error en bloques `try..catch`.

- Para proporcionar funciones de visualización de error exclusivas para errores generados por la aplicación.

  Por ejemplo, se puede crear un nuevo método `toString()` que aplique formato a los mensajes de error de una manera determinada. Asimismo, se puede definir un método `lookupErrorString()` que capture un código de error y recupere el mensaje adecuado según la preferencia de idioma del usuario.

Las clases de error especializadas deben ampliar la clase Error principal de ActionScript. A continuación, se muestra un ejemplo de clase AppError especializada que amplía la clase Error:

```
public class AppError extends Error
{
    public function AppError(message:String, errorID:int)
    {
        super(message, errorID);
    }
}
```

Ejemplo de la utilización de AppError en un proyecto:

```
try
{
    throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
    trace(error.errorID + ": " + error.message)
}
```

Nota: *si se desea sustituir el método* `Error.toString()` *en la subclase, hay que proporcionarle un parámetro . . . (resto). La especificación del lenguaje ECMAScript en la que está basado ActionScript 3.0 define el método* `Error.toString()` *de esta forma, y ActionScript 3.0 lo define del mismo modo para conseguir compatibilidad con versiones anteriores. Por tanto, si se sustituye el método* `Error.toString()`*, los parámetros deben coincidir exactamente. No deben pasarse parámetros al método* `toString()` *en tiempo de ejecución, ya que dichos parámetros se omitirán.*

Respuesta al estado y a los eventos de error

Una de las mejoras más evidentes en la gestión de errores de ActionScript 3.0 es la compatibilidad con la gestión de eventos de error, que permite responder a errores asincrónicos en tiempo de ejecución. (Para obtener una definición de errores asincrónicos, consulte "Tipos de errores" en la página 188.)

Se pueden crear detectores y controladores de eventos para responder a los eventos de error. Muchas clases distribuyen eventos de error igual que otros eventos. Por ejemplo, una instancia de la clase XMLSocket suele distribuir tres tipos de eventos: `Event.CLOSE`, `Event.CONNECT` y `DataEvent.DATA`. No obstante, cuando se produce un problema, la clase XMLSocket puede distribuir los errores `IOErrorEvent.IOError` o `SecurityErrorEvent.SECURITY_ERROR`. Para obtener más información sobre detectores y controladores de eventos, consulte "Gestión de eventos" en la página 254.

Los eventos de error se enmarcan en una de estas dos categorías:

- Eventos de error que amplían la clase ErrorEvent

  La clase flash.events.ErrorEvent contiene las propiedades y los métodos para gestionar los errores en tiempo de ejecución de relativos a las operaciones de red y comunicaciones. Las clases AsyncErrorEvent, IOErrorEvent y SecurityErrorEvent amplían la clase ErrorEvent. Si se usa la versión de depuración de Flash Player o Adobe AIR, un cuadro de diálogo indicará en tiempo de ejecución los eventos de error que encuentre el reproductor sin las funciones de detector.

- Eventos de error basados en estado

  Los eventos de error basados en estado hacen referencia a las propiedades `netStatus` y `status` de las clases de red y comunicaciones. Si Flash Player o Adobe AIR detectan un problema al leer o escribir datos, el valor de las propiedades `netStatus.info.level` o `status.level` (según el objeto de clase que se utilice) se establece en `"error"`. A este error se responde comprobando si la propiedad `level` contiene el valor `"error"` en la función de controlador de eventos.

Trabajo con eventos de error

La clase ErrorEvent y sus subclases contienen tipos de error para gestionar errores distribuidos por Flash Player y Adobe AIR cuando intentan leer o escribir datos.

En el siguiente ejemplo se utiliza una sentencia `try..catch` y controladores de eventos de error para mostrar todos los errores detectados al intentar leer un archivo local. Se puede añadir código de gestión más avanzado a fin de proporcionar opciones a los usuarios o bien gestionar el error automáticamente en los lugares indicados con el comentario "escribir código de gestión de errores aquí":

```
package
{
    import flash.display.Sprite;
    import flash.errors.IOError;
    import flash.events.IOErrorEvent;
    import flash.events.TextEvent;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class LinkEventExample extends Sprite
    {
        private var myMP3:Sound;
        public function LinkEventExample()
        {
            myMP3 = new Sound();
            var list:TextField = new TextField();
            list.autoSize = TextFieldAutoSize.LEFT;
            list.multiline = true;
            list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
            list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
            addEventListener(TextEvent.LINK, linkHandler);
            addChild(list);
        }

        private function playMP3(mp3:String):void
        {
            try
            {
                myMP3.load(new URLRequest(mp3));
                myMP3.play();
            }
            catch (err:Error)
```

```
            {
                trace(err.message);
                // your error-handling code here
            }
            myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
        }

        private function linkHandler(linkEvent:TextEvent):void
        {
            playMP3(linkEvent.text);
            // your error-handling code here
        }

        private function errorHandler(errorEvent:IOErrorEvent):void
        {
            trace(errorEvent.text);
            // your error-handling code here
        }
    }
}
```

Trabajo con eventos de cambio de estado

Flash Player y Adobe AIR cambian dinámicamente el valor de las propiedades `netStatus.info.level` o `status.level` para las clases que admiten la propiedad `level`. Las clases que tienen la propiedad `netStatus.info.level` son NetConnection, NetStream y SharedObject. Las clases que tienen la propiedad `status.level` son HTTPStatusEvent, Camera, Microphone y LocalConnection. Se puede escribir una función de controlador para responder al cambio del valor `level` y hacer un seguimiento de los errores de comunicación.

En el siguiente ejemplo se usa una función `netStatusHandler()` para probar el valor de la propiedad `level`. Si la propiedad `level` indica que se ha detectado un error, el código realiza un seguimiento del mensaje "Video stream failed".

```
package
{
    import flash.display.Sprite;
    import flash.events.NetStatusEvent;
    import flash.events.SecurityErrorEvent;
    import flash.media.Video;
    import flash.net.NetConnection;
    import flash.net.NetStream;

    public class VideoExample extends Sprite
    {
        private var videoUrl:String = "Video.flv";
        private var connection:NetConnection;
        private var stream:NetStream;

        public function VideoExample()
        {
            connection = new NetConnection();
            connection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
            connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR, securityErrorHandler);
            connection.connect(null);
        }
```

```
        private function netStatusHandler(event:NetStatusEvent):void
        {
            if (event.info.level == "error")
            {
                trace("Video stream failed")
            }
            else
            {
                connectStream();
            }
        }

        private function securityErrorHandler(event:SecurityErrorEvent):void
        {
            trace("securityErrorHandler: " + event);
        }

        private function connectStream():void
        {
            var stream:NetStream = new NetStream(connection);
            var video:Video = new Video();
            video.attachNetStream(stream);
            stream.play(videoUrl);
            addChild(video);
        }
    }
}
```

Comparación de las clases Error

ActionScript proporciona varias clases Error predefinidas. Flash Player y Adobe AIR usan muchas de ellas y los usuarios también pueden utilizar las mismas clases Error en su propio código. Existen dos tipos principales de clases Error en ActionScript 3.0: las clases Error principales de ActionScript y las clases Error del paquete flash.error. Las clases Error principales están prescritas en la especificación de lenguaje ECMAScript (ECMA-262) edición 3. El contenido de paquete flash.error está formado por clases adicionales, introducidas para ayudar al desarrollo y la depuración de aplicaciones de ActionScript 3.0.

Clases Error principales de ECMAScript

Entre las clases de error principales de ECMAScript se incluyen Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError y URIError. Cada una de estas clases se encuentra en el espacio de nombres de nivel superior.

Nombre de clase	Descripción	Notas
Error	La clase Error puede usarse para generar excepciones. Se trata de la clase base para otras clases de excepción definidas en ECMAScript: EvalError, RangeError, ReferenceError, SyntaxError, TypeError y URIError.	La clase Error es la clase base para todos los errores en tiempo de ejecución que generan Flash® Player y Adobe® AIR® y se recomienda para cualquier clase de error personalizada.
EvalError	La excepción EvalError se genera si se pasan parámetros al constructor de la clase Function, o bien si el código del usuario llama a la función `eval()`.	En ActionScript 3.0, se ha eliminado la compatibilidad con la función `eval()`; los intentos de usar dicha función generarán errores. Las versiones anteriores de Flash Player utilizaban la función `eval()` para acceder a variables, propiedades, objetos o clips de película por su nombre.
RangeError	Se genera una excepción RangeError si un valor numérico queda fuera del rango admitido.	Por ejemplo, la clase Timer generará una excepción RangeError si la demora es negativa o no finita. También se puede generar RangeError si se intenta añadir un objeto de visualización a una profundidad no válida.
ReferenceError	Se emite una excepción ReferenceError cuando se intenta realizar una referencia a una propiedad no definida en un objeto cerrado (no dinámico). Las versiones del compilador de ActionScript anteriores a ActionScript 3.0 no generaban errores al intentar acceder a una propiedad `undefined`. Sin embargo, ActionScript 3.0 emite la excepción ReferenceError en estas condiciones.	Las excepciones para variables no definidas señalan errores potenciales, lo que ayuda a mejorar la calidad del software. Sin embargo, si el usuario no está acostumbrado a inicializar las variables, es posible que este nuevo comportamiento de ActionScript requiera cambios en sus hábitos de programación.

Nombre de clase	Descripción	Notas
SyntaxError	Se genera una excepción SyntaxError cuando se produce un error de análisis en el código de ActionScript. Para obtener más información, consulte la sección 15.11.6.4 de la especificación del lenguaje ECMAScript (ECMA-262) edición 3 en www.ecma-international.org/publications/standards/Ecma-262.htm. Consulte también la sección 10.3.1 de la especificación ECMAScript for XML (E4X) (ECMA-357 edición 2) en www.ecma-international.org/publications/standards/Ecma-357.htm.	Se puede generar SyntaxError en las circunstancias siguientes: • ActionScript emite excepciones SyntaxError cuando la clase RegExp analiza una expresión normal no válida. • Genera excepciones SyntaxError cuando la clase XMLDocument analiza código XML no válido.
TypeError	Se genera la excepción TypeError cuando el tipo real de un operando es diferente del tipo esperado. Para obtener más información, consulte la sección 15.11.6.5 de la especificación ECMAScript en www.ecma-international.org/publications/standards/Ecma-262.htm, así como la sección 10.3 de la especificación E4X en www.ecma-international.org/publications/standards/Ecma-357.htm.	Se puede generar una excepción TypeError en las circunstancias siguientes: • No se ha podido forzar el tipo de parámetro formal de un parámetro real de una función o un método. • Se ha asignado un valor a una variable y no se puede forzar el tipo de la variable. • El lado derecho del operador `is` o `instanceof` no es un tipo válido. • La palabra clave `super` se utiliza de manera no permitida. • Una consulta de propiedad da como resultado más de una vinculación y, por consiguiente, resulta ambigua. • Se ha invocado un método en un objeto incompatible. Por ejemplo, se genera una excepción TypeError si se inserta un método de la clase RegExp en un objeto genérico y luego se invoca.
URIError	Se genera la excepción URIError cuando una de las funciones de gestión de URI global se utiliza de manera incompatible con esta definición. Para obtener más información, consulte la sección 15.11.6.6 de la especificación ECMAScript en www.ecma-international.org/publications/standards/Ecma-262.htm.	Se puede generar una excepción URIError en las circunstancias siguientes: Se especifica un URI no válido para una función de la API de Flash Player que espera un URI válido, como `Socket.connect()`.

Clases Error principales de ActionScript

Además de las clases Error principales de ECMAScript, ActionScript añade varias clases propias para situaciones de error específicas de ActionScript y funciones de gestión de errores.

Puesto que estas clases son extensiones del lenguaje ActionScript de la especificación del lenguaje ECMAScript (edición 3), potencialmente interesantes como adiciones a una futura versión de la especificación, se mantienen en el nivel superior en lugar de incluirse en un paquete como flash.error.

Nombre de clase	Descripción	Notas
ArgumentError	La clase ArgumentError representa un error que se produce cuando los valores de parámetro proporcionados durante una llamada de función no coinciden con los parámetros definidos para dicha función.	Ejemplos de errores de argumento: • Se proporcionan pocos o demasiados argumentos a un método. • Se esperaba que un argumento fuera miembro de una enumeración, pero no fue así.
SecurityError	La excepción SecurityError se genera cuando se produce una infracción de la seguridad y se deniega el acceso.	Ejemplos de errores de seguridad: • Se realiza un acceso a una propiedad o una llamada a un método no autorizada a través del límite del entorno limitado de seguridad. • Ha habido un intento de acceso a una URL no permitida por el entorno limitado de seguridad. • Se ha intentado establecer una conexión de socket con un puerto, pero la política de sockets necesaria no estaba presente. • Ha habido un intento de acceso a la cámara o al micrófono del usuario; éste ha denegado la petición de acceso al dispositivo.
VerifyError	Se genera una excepción VerifyError cuando se detecta un archivo SWF mal formado o dañado.	Cuando un archivo SWF carga otro archivo SWF, el SWF principal puede capturar una excepción VerifyError generada por el SWF cargado.

Clases Error del paquete flash.error

El paquete flash.error contiene clases Error que se consideran partes integrantes de la API de Flash Player. A diferencia de las clases Error descritas anteriormente, el paquete flash.error comunica eventos de error específicos de Flash Player o Adobe AIR.

Nombre de clase	Descripción	Notas
EOFError	La excepción EOFError se emite al intentar leer más allá del final de los datos disponibles.	Por ejemplo, se emitirá una excepción EOFError si se llama a uno de los métodos de lectura de la interfaz IDataInput y no hay datos suficientes para satisfacer la petición de lectura.
IllegalOperationError	Se genera una excepción IllegalOperationError si un método no se implementa, o bien si la implementación no abarca el uso actual.	Ejemplos de excepciones de errores de operación no permitida: • Una clase base, como DisplayObjectContainer, proporciona mayor funcionalidad de la que puede admitir el escenario. Por ejemplo, si se intenta obtener o establecer una máscara en el escenario (mediante `stage.mask`), Flash Player o Adobe AIR generarán un error IllegalOperationError con el mensaje "La clase Stage no implementa esta propiedad o método". • Una subclase hereda un método que no requiere y que no desea admitir. • Se llama a determinados métodos de accesibilidad al compilar Flash Player sin compatibilidad de accesibilidad. • Las funciones exclusivas de edición se invocan desde una versión de Flash Player en tiempo de ejecución. • Se intenta establecer el nombre de un objeto que se coloca en la línea de tiempo.
IOError	Se genera una excepción IOError cuando se produce algún tipo de excepción de E/S.	Este error se obtiene, por ejemplo, si se intenta realizar una operación de lectura y escritura en un socket no conectado o que haya perdido la conexión.
MemoryError	Se genera una excepción MemoryError cuando falla una petición de asignación de memoria.	De forma predeterminada, la máquina virtual de ActionScript (ActionScript Virtual Machine 2) no impone ningún límite en cuanto a la memoria que puede asignar un programa de ActionScript. En un equipo de escritorio, los errores de asignación de memoria no son frecuentes. Se generará un error cuando el sistema no pueda asignar la memoria necesaria para una operación. De este modo, en un equipo de escritorio, esta excepción es poco habitual, a menos que la petición de asignación sea muy grande. Por ejemplo, una petición de 3.000 millones de bytes resulta imposible, ya que un programa para Microsoft® Windows® de 32 bits sólo puede acceder a 2 GB del espacio de direcciones.
ScriptTimeoutError	Se genera una excepción ScriptTimeoutError cuando se alcanza un intervalo de tiempo de espera del script de 15 segundos. Si se captura una excepción ScriptTimeoutError, se puede gestionar el tiempo de espera del script con mayor comodidad. Si no hay controlador de excepciones, la excepción no capturada mostrará un cuadro de diálogo con un mensaje de error.	Para evitar que un desarrollador malintencionado capture la excepción y permanezca en un bucle infinito, sólo se puede capturar la primera excepción ScriptTimeoutError generada durante la ejecución de un script determinado. El código del usuario no podrá capturar una excepción ScriptTimeoutError posterior; ésta se dirigirá inmediatamente al controlador de excepciones.
StackOverflowError	La excepción StackOverflowError se genera cuando se agota la pila disponible para el script.	Una excepción StackOverflowError puede indicar que se ha producido recursión infinita.

Ejemplo: Aplicación CustomErrors

La aplicación CustomErrors muestra técnicas para trabajar con errores personalizados al crear una aplicación. Estas técnicas son:

- Validar un paquete XML
- Escribir un error personalizado
- Generar errores personalizados
- Notificar un error a los usuarios cuando se genere

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de aplicación CustomErrors pueden encontrarse en la carpeta Samples/CustomError. La aplicación consta de los siguientes archivos:

Archivo	Descripción
CustomErrors.mxml o CustomErrors.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/errors/ApplicationError.as	Una clase que constituye la clase de error base para las clases FatalError y WarningError.
com/example/programmingas3/errors/FatalError.as	Una clase que define un error FatalError que puede generar la aplicación. Esta clase amplía la clase ApplicationError personalizada.
com/example/programmingas3/errors/Validator.as	Una clase que define un método único que valida un paquete XML para empleados proporcionado por el usuario.
com/example/programmingas3/errors/WarningError.as	Una clase que define un error WarningError que puede generar la aplicación. Esta clase amplía la clase ApplicationError personalizada.

Información general sobre la aplicación CustomErrors

Cuando se carga la aplicación, se llama al método `initApp()` en Flex o el código de la línea de tiempo (sin función) se ejecuta en Flash. Este código define un paquete XML de ejemplo que comprobará la clase Validator. Se ejecuta el siguiente código:

```
employeeXML =
    <employee id="12345">
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <costCenter>12345</costCenter>
        <costCenter>67890</costCenter>
    </employee>;
}
```

El paquete XML se muestra después en una instancia de componente TextArea en el escenario, lo que permite modificar el paquete antes de intentar su revalidación.

Cuando el usuario hace clic en el botón Validate, se llama al método `validateData()`. Este método valida el paquete XML para empleados con el método `validateEmployeeXML()` de la clase Validator. El código siguiente muestra el método `validateData()`:

```
function validateData():void
{
    try
    {
        var tempXML:XML = XML(xmlText.text);
        Validator.validateEmployeeXML(tempXML);
        status.text = "The XML was successfully validated.";
    }
    catch (error:FatalError)
    {
        showFatalError(error);
    }
    catch (error:WarningError)
    {
        showWarningError(error);
    }
    catch (error:Error)
    {
        showGenericError(error);
    }
}
```

En primer lugar, se crea un objeto XML temporal con el contenido de la instancia de componente TextArea `xmlText`. A continuación, el método `validateEmployeeXML()` de la clase Validator personalizada (com.example.programmingas3/errors/Validator.as) se invoca y pasa el objeto XML temporal como parámetro. Si el paquete XML es válido, la instancia de componente Label `status` muestra un mensaje de operación realizada correctamente y se cierra la aplicación. Si el método `validateEmployeeXML()` genera un error personalizado (es decir, FatalError, WarningError o Error de tipo genérico), la sentencia `catch` correspondiente llama al método `showFatalError()`, `showWarningError()` o `showGenericError()` y lo ejecuta. Cada uno de estos métodos muestra un mensaje apropiado en un área de texto denominada `statusText` para notificar al usuario el error concreto que se ha producido. Los métodos también actualizan la instancia de componente Label `status` con un mensaje determinado.

Si se produce un error grave durante el intento de validar el paquete XML para empleados, el mensaje de error se muestra en el área de texto `statusText` y el componente TextArea `xmlText` y la instancia del componente Button `validateBtn` se deshabilitan, tal como se indica en el código siguiente:

```
function showFatalError(error:FatalError):void
{
    var message:String = error.message + "\n\n";
    var title:String = error.getTitle();
    statusText.text = message + " " + title + "\n\nThis application has ended.";
    this.xmlText.enabled = false;
    this.validateBtn.enabled = false;
    hideButtons();
}
```

Si se produce un error de advertencia en lugar de un error grave, el mensaje de error se muestra en la instancia de TextArea `statusText`, pero las instancias del componente TextField y Button `xmlText` no se deshabilitan. El método `showWarningError()` muestra el mensaje de error personalizado en el área de texto `statusText`. El mensaje también pregunta al usuario si desea continuar con la validación del XML o cancelar la ejecución del script. El siguiente fragmento de código muestra el método `showWarningError()`:

```
function showWarningError(error:WarningError):void
{
    var message:String = error.message + "\n\n" + "Do you want to exit this application?";
    showButtons();
    var title:String = error.getTitle();
    statusText.text = message;
}
```

Cuando el usuario hace clic en el botón Sí o No, se invoca el método`closeHandler()`. El extracto siguiente muestra el método `closeHandler()`:

```
function closeHandler(event:CloseEvent):void
{
    switch (event.detail)
    {
        case yesButton:
            showFatalError(new FatalError(9999));
            break;
        case noButton:
            statusText.text = "";
            hideButtons();
            break;
    }
}
```

Si el usuario decide cancelar la ejecución del script haciendo clic en Sí, se genera un error de tipo FatalError, lo que provoca el cierre de la aplicación.

Creación de un validador personalizado

La clase Validator personalizada contiene un solo método, `validateEmployeeXML()`. El método `validateEmployeeXML()` sólo procesa un único argumento, `employee`, que es el paquete XML que se desea validar. El método `validateEmployeeXML()` funciona del modo siguiente:

```
public static function validateEmployeeXML(employee:XML):void
{
    // checks for the integrity of items in the XML
    if (employee.costCenter.length() < 1)
    {
        throw new FatalError(9000);
    }
    if (employee.costCenter.length() > 1)
    {
        throw new WarningError(9001);
    }
    if (employee.ssn.length() != 1)
    {
        throw new FatalError(9002);
    }
}
```

Para que se valide un empleado, éste debe pertenecer única y exclusivamente a un centro de coste. Si el empleado no pertenece a ningún centro de coste, el método genera un error de tipo FatalError, que se propaga hasta el método `validateData()` del archivo de aplicación principal. Si el empleado pertenece a más de un centro de coste, se genera un error WarningError. La última comprobación en el validador de XML consiste en confirmar que el usuario tiene exactamente un número de seguridad social definido (el nodo `ssn` del paquete XML). Si no hay exactamente un nodo `ssn`, se genera un error de tipo FatalError.

Se pueden añadir otras comprobaciones al método `validateEmployeeXML()`; por ejemplo, para asegurarse de que el nodo `ssn` contiene un número válido, o bien el empleado tiene definidos al menos un número de teléfono y una dirección de correo electrónico, y los dos valores son válidos. Asimismo, se pueden modificar los datos XML para que cada empleado tenga un ID único y especifique el ID de su responsable.

Definición de la clase ApplicationError

La clase ApplicationError es la clase base para las clases FatalError y WarningError. La clase ApplicationError amplía la clase Error y define sus propios métodos y propiedades, entre los que se incluye la definición de ID y gravedad de errores, así como objetos XML que contienen códigos y mensajes de error personalizados. Esta clase también define dos constantes estáticas que se usan para definir la gravedad de cada tipo de error.

El método del constructor de la clase ApplicationError es el siguiente:

```
public function ApplicationError()
{
    messages =
        <errors>
            <error code="9000">
                <![CDATA[Employee must be assigned to a cost center.]]>
            </error>
            <error code="9001">
                <![CDATA[Employee must be assigned to only one cost center.]]>
            </error>
            <error code="9002">
                <![CDATA[Employee must have one and only one SSN.]]>
            </error>
            <error code="9999">
                <![CDATA[The application has been stopped.]]>
            </error>
        </errors>;
}
```

Cada nodo de error del objeto XML contiene un código numérico único y un mensaje de error. Los mensajes de error pueden consultarse por su código de error mediante E4X, tal como se muestra en el método `getMessageText()` siguiente:

```
public function getMessageText(id:int):String
{
    var message:XMLList = messages.error.(@code == id);
    return message[0].text();
}
```

El método `getMessageText()` procesa un solo argumento de tipo entero, `id`, y devuelve una cadena. El argumento `id` es el código de error que debe consultar el error. Por ejemplo, al pasar el `id` 9001, se recupera un error que indica que los empleados deben asignarse a un solo centro de coste. Si hay más de un error con el mismo código, ActionScript sólo devuelve el mensaje de error para el primer resultado encontrado (`message[0]` del objeto XMLList devuelto).

El siguiente método de esta clase, `getTitle()`, no procesa ningún parámetro y devuelve un valor de cadena que contiene el ID de este error específico. Este valor se emplea para ayudar a identificar fácilmente el error exacto, producido durante la validación del paquete XML. El siguiente fragmento de código muestra el método `getTitle()`:

```
public function getTitle():String
{
    return "Error #" + id;
}
```

El último método de la clase ApplicationError es `toString()`. Este método sustituye la función definida en la clase Error, de manera que se puede personalizar la presentación del mensaje de error. El método devuelve una cadena que identifica el mensaje y el número de error específicos que se han producido.

```
public override function toString():String
{
    return "[APPLICATION ERROR #" + id + "] " + message;
}
```

Definición de la clase FatalError

La clase FatalError amplía la clase personalizada ApplicationError y define tres métodos: el contructor FatalError, `getTitle()` y `toString()`. El primer método (el constructor de FatalError) procesa un solo argumento de tipo entero (`errorID`), y establece la gravedad del error con los valores de constantes estáticas definidas en la clase ApplicationError. El mensaje de error específico se obtiene llamando al método `getMessageText()` de la clase ApplicationError. El constructor de FatalError es el siguiente:

```
public function FatalError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.FATAL;
    message = getMessageText(errorID);
}
```

El siguiente método de la clase FatalError, `getTitle()`, sustituye el método `getTitle()` definido anteriormente en la clase ApplicationError, y añade el texto "-- FATAL" al título para informar al usuario de que se ha producido un error grave. El método `getTitle()` funciona del modo siguiente:

```
public override function getTitle():String
{
    return "Error #" + id + " -- FATAL";
}
```

El último método de esta clase, `toString()`, sustituye el método `toString()` definido en la clase ApplicationError. El método `toString()` funciona es:

```
public override function toString():String
{
    return "[FATAL ERROR #" + id + "] " + message;
}
```

Definición de la clase WarningError

La clase WarningError amplía la clase ApplicationError y es casi idéntica a la clase FatalError, salvo por un par de cambios en cadenas poco importantes. La gravedad de los errores se establece en ApplicationError.WARNING en lugar de ApplicationError.FATAL, tal como se muestra en el código siguiente:

```
public function WarningError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.WARNING;
    message = super.getMessageText(errorID);
}
```

Capítulo 10: Utilización de expresiones regulares

Una expresión regular describe un patrón que se utiliza para buscar y manipular texto coincidente en cadenas. Las expresiones regulares parecen cadenas, pero pueden incluir códigos especiales para describir patrones y repeticiones. Por ejemplo, la siguiente expresión regular detecta una cadena que empiece por el carácter A seguido de uno o más dígitos:

```
/A\d+/
```

En este capítulo se describe la sintaxis básica para crear expresiones regulares. Sin embargo, las expresiones regulares pueden tener muchas complejidades y matices. Se puede encontrar información detallada sobre expresiones regulares en Internet y en librerías. Hay que tener en cuenta que los distintos entornos de programación implementan las expresiones regulares de distinta manera. ActionScript 3.0 implementa la definición de las expresiones regulares incluida en la especificación del lenguaje ECMAScript edición 3 (ECMA-262).

Fundamentos de la utilización de expresiones regulares

Introducción a la utilización de expresiones regulares

Una expresión regular describe un patrón de caracteres. Las expresiones regulares se suelen utilizar para comprobar que un valor de texto se ajusta a un patrón específico (por ejemplo, para comprobar que un número de teléfono especificado por el usuario tiene el número de dígitos correcto) o para sustituir partes de un valor de texto que coincidan con un patrón específico.

Las expresiones regulares pueden ser sencillas. Por ejemplo, se puede desear confirmar que una cadena específica coincide con "ABC" o reemplazar cada instancia de "ABC" en una cadena por otro texto. En este caso, se puede utilizar la siguiente expresión regular, que define el patrón formado por las letras A, B y C en secuencia:

```
/ABC/
```

Hay que tener en cuenta que el literal de expresión regular se delimita con el carácter barra diagonal (`/`).

Los patrones de expresión regular también pueden ser complejos y, a veces, aparentemente crípticos, como la siguiente expresión para detectar una dirección de correo electrónico válida:

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

Normalmente las expresiones regulares se utilizan para buscar patrones en cadenas y para reemplazar caracteres. En estos casos, se creará un objeto de expresión regular y se utilizará como parámetro de uno de varios métodos de la clase String. Los siguientes métodos de la clase String toman como parámetros expresiones regulares: `match()`, `replace()`, `search()` y `split()`. Para más información sobre estos métodos, consulte "Búsqueda de patrones en cadenas y sustitución de subcadenas" en la página 150

La clase RegExp incluye los métodos siguientes: `test()` y `exec()`. Para más información, consulte "Métodos para utilizar expresiones regulares con cadenas" en la página 226.

Tareas comunes con expresiones regulares

A continuación se muestran algunos usos comunes de las expresiones regulares, que se describen en mayor detalle en este capítulo:

- Crear un patrón de expresión regular
- Utilizar caracteres especiales en patrones
- Identificar secuencias de varios caracteres (como "un número de dos dígitos" o "entre siete y diez letras")
- Identificar cualquier carácter de un rango de letras o números (como "cualquier letra entre *a* y *m*")
- Identificar un carácter de un conjunto de caracteres posibles
- Identificar subsecuencias (segmentos de un patrón)
- Detectar y reemplazar texto utilizando patrones

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Carácter de escape: carácter que indica que el carácter que sigue debe tratarse como un metacarácter en lugar de como un carácter literal. En sintaxis de expresiones regulares, el carácter de barra diagonal inversa (\) es el carácter de escape, por lo que una barra diagonal inversa seguida de otro carácter se interpretará como un código especial en lugar de interpretarse literalmente.
- Indicador: carácter que especifica alguna opción sobre cómo debe utilizarse el patrón de expresión regular (p. ej., distinguir entre caracteres en mayúsculas y caracteres en minúsculas.
- Metacarácter: carácter que tiene un significado especial en un patrón de expresión regular, en lugar de representar literalmente dicho carácter en el patrón.
- Cuantificador: carácter (o conjunto de caracteres) que indica cuántas veces debe repetirse una parte del patrón. Por ejemplo, se puede utilizar un cuantificador para especificar que un código postal de Estados Unidos debe contener cinco o nueve números.
- Expresión regular: sentencia de programa que define un patrón de caracteres que se puede utilizar para confirmar si otras cadenas coinciden con ese patrón o para sustituir partes de una cadena.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Como los listados de código de este capítulo consisten principalmente en patrones de expresiones regulares, para probar los ejemplos hay que realizar unos cuantos pasos:

1. Cree un documento de Flash nuevo.
2. Seleccione un fotograma clave y abra el panel Acciones.
3. Cree una variable RegExp (expresión regular) como ésta:

```
var pattern:RegExp = /ABC/;
```

4. Copie el patrón del ejemplo y asígnele el valor de la variable RegExp. Por ejemplo, en la línea de código anterior, el patrón es la parte del código situada a la derecha del signo igual, sin incluir el signo de punto y coma (`/ABC/`).
5. Cree una o más variables String que contengan cadenas apropiadas para probar la expresión regular. Por ejemplo, si va a crear una expresión regular para validar direcciones de correo electrónico, cree unas cuantas variables String que contengan direcciones de correo electrónico válidas y otras cuantas que no sean válidas:

```
var goodEmail:String = "bob@example.com";
var badEmail:String = "5@$2.99";
```

6 Añada líneas de código para probar las variables String y determinar si coinciden con el patrón de la expresión regular. Éstos serán los valores que deseará mostrar en pantalla mediante la función `trace()` o escribiéndolos en un campo de texto en el escenario.

```
trace(goodEmail, " is valid:", pattern.test(goodEmail));
trace(badEmail, " is valid:", pattern.test(badEmail));
```

Por ejemplo, suponiendo que `pattern` define el patrón de la expresión regular para una dirección de correo electrónico válida, las líneas de código anteriores escriben este texto en el panel Salida:

```
bob@example.com is valid: true
5@$2.99 is valid: false
```

Para más información sobre cómo probar valores escribiéndolos en una instancia de campo de texto en el escenario o utilizando la función `trace()` para imprimir los valores en el panel Salida, consulte "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

Sintaxis de las expresiones regulares

En esta sección se describen todos los elementos de sintaxis de las expresiones regulares de ActionScript. Sin embargo, las expresiones regulares pueden tener muchas complejidades y matices. Se puede encontrar información detallada sobre expresiones regulares en Internet y en librerías. Hay que tener en cuenta que los distintos entornos de programación implementan las expresiones regulares de distinta manera. ActionScript 3.0 implementa la definición de las expresiones regulares incluida en la especificación del lenguaje ECMAScript edición 3 (ECMA-262).

Generalmente, se utilizan expresiones regulares que detectan patrones más complicados que una simple cadena de caracteres. Por ejemplo, la siguiente expresión regular define el patrón formado por la secuencia de letras A, B y C seguida de un dígito:

```
/ABC\d/
```

El código `\d` representa "cualquier dígito". El carácter barra diagonal inversa (`\`) se denomina carácter de escape y, combinado con el carácter siguiente (en este caso, la letra d), tiene un significado especial en la expresión regular. En este capítulo se describen estas secuencias de carácter de escape y otras características de la sintaxis de las expresiones regulares.

La siguiente expresión regular define el patrón de las letras ABC seguidas de un número arbitrario de dígitos (obsérvese el asterisco):

```
/ABC\d*/
```

El asterisco (`*`) es un *metacarácter*. Un metacarácter es un carácter que tiene un significado especial en las expresiones regulares. El asterisco es un tipo específico de metacarácter denominado *cuantificador* que se utiliza para cuantificar la cantidad de repeticiones de un carácter o grupo de caracteres. Para más información, consulte "Cuantificadores" en la página 218.

Además del patrón, una expresión regular puede contener indicadores, que especifican cómo debe detectarse la expresión regular. Por ejemplo, la siguiente expresión regular utiliza el indicador `i`, que especifica que la expresión regular no distinguirá mayúsculas de minúsculas en las cadenas coincidentes:

```
/ABC\d*/i
```

Para más información, consulte "Indicadores y propiedades" en la página 223.

Se pueden utilizar expresiones regulares con los siguientes métodos de la clase String: `match()`, `replace()` y `search()`. Para más información sobre estos métodos, consulte "Búsqueda de patrones en cadenas y sustitución de subcadenas" en la página 150

Creación de una instancia de expresión regular

Hay dos maneras de crear una instancia de expresión regular. Una consiste en utilizar caracteres de barra diagonal (`/`) para delimitar la expresión regular; la otra utiliza el constructor `new`. Por ejemplo, las dos expresiones regulares siguientes son equivalentes:

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

Las barras diagonales delimitan un literal de expresión regular de la misma manera que las comillas delimitan un literal de cadena. La parte de la expresión regular delimitada por las barras diagonales define el *patrón*. La expresión regular también puede incluir *indicadores* a continuación de la última barra de delimitación. Se considera que estos indicadores forman parte de la expresión regular, pero son independientes del patrón.

Al utilizar el constructor `new` se pueden utilizar dos cadenas para definir la expresión regular. La primera cadena define el patrón y la segunda cadena define los indicadores, como en el siguiente ejemplo:

```
var pattern2:RegExp = new RegExp("bob", "i");
```

Si se incluye una barra diagonal *en* una expresión regular definida con los delimitadores de barra diagonal, hay que escribir inmediatamente antes de dicha barra diagonal el carácter de escape de barra diagonal inversa (`\`). Por ejemplo, la siguiente expresión regular detecta el patrón `1/2`:

```
var pattern:RegExp = /1\/2/;
```

Para incluir comillas *en* una expresión regular definida con el constructor `new`, hay que añadir el carácter de escape de barra diagonal inversa (`\`) antes de las comillas (de la misma manera que al definir un literal de cadena). Por ejemplo, las siguientes expresiones regulares detectan el patrón `eat at "joe's"`:

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

No se debe utilizar el carácter de escape barra diagonal inversa con comillas en expresiones regulares definidas con los delimitadores de barra diagonal. De forma similar, no se debe utilizar el carácter de escape con barras diagonales en expresiones regulares definidas con el constructor `new`. Las siguientes expresiones regulares son equivalentes y definen el patrón `1/2 "joe's"`:

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

Además, en una expresión definida con el constructor `new`, para utilizar una metasecuencia que comience con el carácter barra diagonal inversa (`\`) como, por ejemplo, `\d` (que coincide con cualquier dígito), debe escribirse el carácter barra diagonal inversa dos veces:

```
var pattern:RegExp = new RegExp("\\d+", ""); // matches one or more digits
```

En este caso hay que escribir el carácter barra diagonal inversa dos veces, ya que el primer parámetro del método constructor `RegExp()` es una cadena y en un literal de cadena hay que escribir un carácter barra diagonal inversa dos veces para que se reconozca como un solo carácter barra diagonal inversa.

En las secciones siguientes se describe la sintaxis para definir patrones de expresiones regulares.

Para más información sobre los indicadores, consulte "Indicadores y propiedades" en la página 223.

Caracteres, metacaracteres y metasecuencias

La expresión regular más sencilla es la que detecta una secuencia de caracteres, como en el siguiente ejemplo:

```
var pattern:RegExp = /hello/;
```

No obstante, los siguientes caracteres, denominados metacaracteres, tienen significados especiales en las expresiones regulares:

```
^ $ \ . * + ? ( ) [ ] { } |
```

Por ejemplo, la siguiente expresión regular detecta la letra A seguida de cero o más instancias de la letra B (el metacarácter asterisco indica esta repetición), seguidas de la letra C:

```
/AB*C/
```

Para incluir un metacarácter sin su significado especial en un patrón de expresión regular, hay que utilizar el carácter de escape de barra diagonal inversa (\). Por ejemplo, la siguiente expresión regular detecta la letra A seguida de la letra B, seguida de un asterisco, seguido de la letra C:

```
var pattern:RegExp = /AB\*C/;
```

Una *metasecuencia*, al igual que un metacarácter, tiene un significado especial en una expresión regular. Las metasecuencias están formadas por más de un carácter. Las secciones siguientes proporcionan detalles sobre la utilización de metacaracteres y metasecuencias.

Metacaracteres

En la tabla siguiente se muestra un resumen de los metacaracteres que se pueden utilizar en las expresiones regulares:

Metacarácter	Descripción
^ (intercalación)	Detecta el principio de la cadena. Con el indicador `m` (`multiline`) establecido, el signo de intercalación también detecta el inicio de cada línea (consulte "Indicadores y propiedades" en la página 223). Hay que tener en cuenta que cuando se utiliza al principio de una clase de caracteres, el signo de intercalación indica negación, no el principio de una cadena. Para más información, consulte "Clases de caracteres" en la página 217.
$ (signo dólar)	Detecta el final de la cadena. Con el indicador `m` (`multiline`) establecido, `$` detecta la posición anterior a un carácter de nueva línea (`\n`). Para más información, consulte "Indicadores y propiedades" en la página 223.
\ (barra diagonal inversa)	Omite el significado especial de los metacaracteres. También debe utilizarse el carácter barra diagonal inversa si se desea utilizar un carácter barra diagonal en un literal de expresión regular, como `/1\/2/` (para detectar el carácter 1, seguido del carácter barra diagonal, seguido del carácter 2).
. (punto)	Detecta cualquier carácter individual. El punto detecta un carácter de nueva línea (`\n`) sólo si está establecido el indicador `s` (`dotall`). Para más información, consulte "Indicadores y propiedades" en la página 223.
* (asterisco)	Detecta el elemento anterior repetido cero o más veces. Para más información, consulte "Cuantificadores" en la página 218.
+ (más)	Detecta el elemento anterior repetido una o más veces. Para más información, consulte "Cuantificadores" en la página 218.
? (signo de interrogación)	Detecta el elemento anterior repetido cero veces o una sola vez. Para más información, consulte "Cuantificadores" en la página 218.

Metacarácter	Descripción
`(` y `)`	Define grupos dentro de la expresión regular. Los grupos se pueden utilizar para: • Para limitar el ámbito del alternador |: `/(a|b|c)d/` • Para definir el ámbito de un cuantificador: `/(walla.){1,2}/` • En referencias a elementos detectados previamente. Por ejemplo, en la siguiente expresión regular, `\1` detectará lo que se haya detectado en el primer grupo delimitado con paréntesis del patrón: • `/(\w*) se repite: \1/` Para más información, consulte "Grupos" en la página 220.
`[` y `]`	Define una clase de caracteres, que especifica posibles coincidencias para un solo carácter: `/[aeiou]/` detecta cualquiera de los caracteres especificados. En las clases de caracteres se utiliza un guión (-)para designar un rango de caracteres: `/[A-Z0-9]/` detecta cualquier letra mayúscula (A a Z) o cualquier dígito (0 a 9). Además, se debe insertar una barra diagonal inversa para omitir el significado especial de los caracteres] y - caracteres: `/[+\-]\d+/` detecta + o - antes de uno o más dígitos. En las clases de caracteres, los caracteres que normalmente metacaracteres se tratan como caracteres normales (no metacaracteres), sin necesidad de utilizar una barra diagonal inversa: `/[$]/`£ detecta $o £. Para más información, consulte "Clases de caracteres" en la página 217.
| *(barra vertical)*	Se utiliza para la alternancia, a fin de detectar la parte de la izquierda o la parte de la derecha: `/abc|xyz/` detecta `abc` o `xyz`.

Metasecuencias

Las metasecuencias son secuencias de caracteres que tienen un significado especial en un patrón de expresión regular. En la tabla siguiente se describen estas metasecuencias:

Metasecuencia	Descripción
{***n***} {***n***,} y {***n***,***n***}	Especifica un cuantificador numérico o un rango de cuantificadores para el elemento anterior: `/A{27}/` detecta el carácter `A` repetido `27` veces. `/A{3,}/` detecta el carácter `A` repetido `3` o más veces. `/A{3,5}/` detecta el carácter `A` repetido entre `3` y `5` veces. Para más información, consulte "Cuantificadores" en la página 218.
`\b`	Detecta la posición entre un carácter de palabra y un carácter de otro tipo. Si el primer o el último carácter de la cadena es un carácter de palabra, también detecta el principio o el final de la cadena.
`\B`	Detecta la posición entre dos caracteres de palabra. También detecta la posición entre dos caracteres de otro tipo.
`\d`	Detecta un dígito decimal.
`\D`	Detecta cualquier carácter que no sea un dígito.
`\f`	Detecta un carácter de salto de página.
`\n`	Detecta el carácter de nueva línea.

Metasecuencia	Descripción
\r	Detecta el carácter de retorno.
\s	Detecta cualquier carácter de espacio en blanco (un carácter de espacio, tabulación, nueva línea o retorno).
\S	Detecta cualquier carácter que no sea un espacio en blanco.
\t	Detecta el carácter de tabulación.
\u*nnnn*	Detecta el carácter Unicode con el código de carácter especificado por el número hexadecimal *nnnn*. Por ejemplo, \u263a es el carácter smiley.
\v	Detecta un carácter de avance vertical.
\w	Detecta un carácter de palabra (A–Z, a–z, 0–9 o _). Hay que tener en cuenta que \w no detecta caracteres que no sean los del idioma inglés, como é, ñ o ç .
\W	Detecta cualquier carácter que no sea un carácter de palabra.
\\x*nn*	Detecta el carácter con el valor ASCII especificado, definido por el número hexadecimal *nn*.

Clases de caracteres

Se pueden utilizar clases de caracteres para especificar una lista de caracteres con el fin de detectar una posición en la expresión regular. Las clases de caracteres se definen con corchetes ([y]). Por ejemplo, la siguiente expresión regular define una clase de caracteres que detecta bag, beg, big, bog o bug:

```
/b[aeiou]g/
```

Secuencias de escape en clases de caracteres

La mayoría de los metacaracteres y las metasecuencias que normalmente tienen significados especiales en una expresión regular *no* tienen el mismo significado dentro de una clase de caracteres. Por ejemplo, en una expresión regular, el asterisco se utiliza para indicar repetición, pero en una clase de caracteres no tiene este significado. La siguiente clase de caracteres detecta el asterisco literalmente, junto con cualquiera de los otros caracteres indicados:

```
/[abc*123]/
```

Sin embargo, los tres caracteres mostrados en la tabla siguiente funcionan como metacaracteres (tienen un significado especial) en las clases de caracteres:

Metacarácter	Significado en las clases de caracteres
]	Define el final de la clase de caracteres.
-	Define un rango de caracteres (consulte la siguiente sección "Rangos de caracteres en clases de caracteres").
\	Define metasecuencias y omite el significado especial de los metacaracteres.

Para que se reconozca cualquiera de estos caracteres como un carácter literal (sin su significado especial de metacarácter), hay que escribir el carácter de escape barra diagonal inversa inmediatamente antes del carácter. Por ejemplo, la siguiente expresión regular incluye una clase de caracteres que detecta cualquiera de los cuatro símbolos ($, \,] o -):

```
/[$\\\]\-]/
```

Además de los metacaracteres que conservan su significado especial, las siguientes metasecuencias funcionan como metasecuencias en las clases de caracteres:

Metasecuencia	Significado en las clases de caracteres
`\n`	Detecta un carácter de nueva línea.
`\r`	Detecta un carácter de retorno.
`\t`	Detecta un carácter de tabulación.
`\u`***nnnn***	Detecta el carácter con el valor de código Unicode especificado (definido por el número hexadecimal *nnnn*).
`\\x`***nn***	Detecta el carácter con el valor ASCII especificado (definido por el número hexadecimal *nn*).

Otros metacaracteres y metasecuencias de expresión regular se tratan como caracteres normales dentro de una clase de caracteres.

Rangos de caracteres en clases de caracteres

Se utiliza un guión para especificar un rango de caracteres, como `A-Z`, `a-z` o `0-9`. Estos caracteres deben constituir un rango válido en el conjunto de caracteres. Por ejemplo, la siguiente clase de caracteres detecta cualquiera de los caracteres del intervalo `a-z` o cualquier dígito:

```
/[a-z0-9]/
```

También se puede utilizar el código de carácter ASCII `\\x`*nn* para especificar un rango por valor ASCII. Por ejemplo, la siguiente clase de caracteres detecta cualquier carácter de un conjunto de caracteres extendidos ASCII (como é y ê):

```
\\x
```

Clases de caracteres denegados

Si se utiliza un carácter de intercalación (`^`) al principio de una clase de caracteres, deniega los caracteres de la clase: detectará cualquier carácter que no esté en la clase. La siguiente clase de caracteres detecta cualquier carácter *salvo* una letra minúscula (`a–z–`) o un dígito:

```
/[^a-z0-9]/
```

Para indicar la negación se debe escribir el carácter de intercalación (`^`) al *principio* de una clase de caracteres. De lo contrario, sólo se añade el carácter de intercalación a los caracteres de la clase de caracteres. Por ejemplo, la siguiente clase de caracteres detecta cualquier carácter de un conjunto de caracteres de símbolo, incluido el signo de intercalación:

```
/[!.,#+*%$&^]/
```

Cuantificadores

Los cuantificadores se utilizan para especificar repeticiones de caracteres o secuencias en patrones, de la manera siguiente:

Metacarácter de cuantificador	Descripción
* (asterisco)	Detecta el elemento anterior repetido cero o más veces.

Metacarácter de cuantificador	Descripción
+ (más)	Detecta el elemento anterior repetido una o más veces.
? (signo de interrogación)	Detecta el elemento anterior repetido cero veces o una sola vez.
{*n*} {*n*,} y {*n*,*n*}	Especifica un cuantificador numérico o un rango de cuantificadores para el elemento anterior: /A{27}/ detecta el carácter A repetido 27 veces. /A{3,}/ detecta el carácter A repetido 3 o más veces. /A{3,5}/ detecta el carácter A repetido entre 3 y 5 veces.

Se puede aplicar un cuantificador a un solo carácter, a una clase de caracteres o a un grupo:

- `/a+/` detecta el carácter `a` repetido una o más veces.
- `/\d+/` detecta uno o más dígitos.
- `/[abc]+/` detecta una repetición de uno o más caracteres, cada uno de las cuales puede sera, b o c.
- `/(very, )*/` detecta la palabra `very` seguida de una coma y un espacio repetido cero o más veces.

Se pueden utilizar cuantificadores dentro de grupos delimitados por paréntesis que tengan cuantificadores aplicados. Por ejemplo, el siguiente cuantificador detecta cadenas como `word` y `word-word-word`:

```
/\w+(-\w+)*/
```

De manera predeterminada, las expresiones regulares realizan lo que se conoce como una *detección de la coincidencia más larga posible (greedy matching)*. Cualquier subpatrón de la expresión regular (como `.*`) intentará detectar la mayor cantidad posible de caracteres en la cadena antes de pasar a la siguiente parte de la expresión regular. Por ejemplo, considérense la expresión regular y la cadena siguientes:

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

La expresión regular detecta toda la cadena:

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

Sin embargo, si sólo se desea detectar una agrupación `<p>...</p>`, se puede hacer lo siguiente:

```
<p>Paragraph 1</p>
```

Añadir un signo de interrogación (`?`) a continuación de cualquier cuantificador para convertirlo en un cuantificador *perezoso*, que detecta la coincidencia más corta posible. Por ejemplo, la siguiente expresión regular, que utiliza un cuantificador de este tipo, `*?` , detecta `<p>` seguido del mínimo número posible de caracteres, seguidos de `</p>`:

```
/<p>.*?<\/p>/
```

Hay que tener en cuenta los siguientes aspectos sobre los cuantificadores:

- Los cuantificadores `{0}` y `{0,0}` no excluyen un elemento de una coincidencia.
- No se deben combinar varios cuantificadores, como en `/abc+*/`.
- El punto (.) no abarca líneas a menos que se establezca el indicador `s` (`dotall`), aunque esté seguido de un cuantificador `*`. Por ejemplo, considérese el fragmento de código siguiente:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
    // output: <p>Test
    //                 Multiline</p>
```

Para más información, consulte "Indicadores y propiedades" en la página 223.

Alternancia

El carácter | (barra vertical) se utiliza en una expresión regular para que el motor de expresiones regulares considere alternativas para la detección. Por ejemplo, la siguiente expresión regular detecta cualquiera de las palabras `cat, dog, pig, rat`:

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

Se pueden utilizar paréntesis para definir grupos a fin de restringir el ámbito del alternador |. La siguiente expresión regular detecta `cat` seguida de `nap` o `nip`:

```
var pattern:RegExp = /cat(nap|nip)/;
```

Para más información, consulte "Grupos" en la página 220.

Las dos expresiones regulares siguientes, una con el alternador | y la otra con una clase de caracteres (definida con `[` y `]`), son equivalentes:

```
/1|3|5|7|9/
/[13579]/
```

Para más información, consulte "Clases de caracteres" en la página 217.

Grupos

Se puede especificar un grupo en una expresión regular utilizando paréntesis, de la manera siguiente:

```
/class-(\d*)/
```

Un grupo es una subsección de un patrón. Los grupos se pueden utilizar para:

- Aplicar un cuantificador a más de un carácter.
- Delimitar subpatrones que debe aplicarse mediante alternancia (utilizando el carácter |).
- Capturar coincidencias de subcadenas (por ejemplo, utilizando `\1` en una expresión regular para detectar un grupo detectado previamente o utilizando `$1` de forma similar en el método `replace()` de la clase String).

En las secciones siguientes se proporcionan detalles sobre estos usos de los grupos.

Utilización de grupos con cuantificadores

Si no se utiliza un grupo, un cuantificador se aplica al carácter o la clase de caracteres que lo precede, como se indica a continuación:

```
var pattern:RegExp = /ab*/ ;
// matches the character a followed by
// zero or more occurrences of the character b

pattern = /a\d+/;
// matches the character a followed by
// one or more digits

pattern = /a[123]{1,3}/;
// matches the character a followed by
// one to three occurrences of either 1, 2, or 3
```

No obstante, se puede utilizar un grupo para aplicar un cuantificador a más de un carácter o clase de caracteres:

```
var pattern:RegExp = /(ab)*/;
// matches zero or more occurrences of the character a
// followed by the character b, such as abababab

pattern = /(a\d)+/;
// matches one or more occurrences of the character a followed by
// a digit, such as a1a5a8a3

pattern = /(spam ){1,3}/;
// matches 1 to 3 occurrences of the word spam followed by a space
```

Para más información sobre los cuantificadores, consulte "Cuantificadores" en la página 218.

Utilización de los grupos con el carácter alternador (|)

Se pueden utilizar grupos para definir el grupo de caracteres al que se desea aplicar un carácter alternador (`|`), de la manera siguiente:

```
var pattern:RegExp = /cat|dog/;
// matches cat or dog

pattern = /ca(t|d)og/;
// matches catog or cadog
```

Utilización de grupos para capturar coincidencias de subcadenas

Si se define un grupo delimitado por paréntesis estándar en un patrón, se puede hacer referencia al mismo en una parte posterior de la expresión regular. Esto se denomina *referencia a un elemento detectado previamente (backreference)* y estos tipos de grupos se denominan *grupos de captura*. Por ejemplo, en la siguiente expresión regular, la secuencia `\1` detectará la subcadena que se haya detectado en el primer grupo de captura delimitado con paréntesis:

```
var pattern:RegExp = /(\d+)-by-\1/;
// matches the following: 48-by-48
```

Se pueden especificar hasta 99 de estas referencias a elementos detectados previamente escribiendo `\1`, `\2`, ..., `\99`.

De forma similar, en el método `replace()` de la clase String, se puede utilizar `$1$99`– para insertar subcadenas coincidentes detectadas con un grupo de captura en la cadena de sustitución:

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
    // output: Bob, hello.
```

Además, si se utilizan grupos de captura, el método `exec()` de la clase RegExp y el método `match()` de la clase String devuelven subcadenas que detectan los grupos de captura:

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example,com
```

Utilización de grupos que no capturan y grupos de búsqueda hacia delante

Un grupo que no captura es un grupo que sólo se utiliza para agrupar; no se "almacena" ni proporciona referencias numeradas de elementos detectados previamente. Para definir grupos que no capturan se utiliza `(?: y )`, de la manera siguiente:

```
var pattern = /(?:com|org|net);
```

Por ejemplo, véase la diferencia entre colocar `(com|org)` en grupo de captura y en un grupo que no captura (el método `exec()` muestra los grupos de captura después de la coincidencia completa):

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
// bob@example.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example
```

Un tipo especial de grupo que no captura es el *grupo de búsqueda hacia delante*, del cual hay dos tipos: el *grupo de búsqueda positiva hacia delante* y el *grupo de búsqueda negativa hacia delante*.

Para definir un grupo de búsqueda positiva hacia delante, que especifica que el subpatrón del grupo debe coincidir en la posición, se utiliza `(?=` y `)`. No obstante, la parte de la cadena que coincide con el grupo de búsqueda positiva hacia delante puede coincidir con los demás patrones de la expresión regular. Por ejemplo, como `(?=e)` es un grupo de búsqueda positiva hacia delante en el código siguiente, el carácter `e` que detecta puede ser detectado por una parte posterior de la expresión regular, en este caso, el grupo de captura, `\w*)`:

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

Para definir un grupo de búsqueda negativa hacia delante, que especifica que el subpatrón del grupo `no` debe coincidir en la posición, se utiliza `(?!` y *)*. Por ejemplo:

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

Utilización de grupos con nombre

Un grupo con nombre es un tipo de grupo en una expresión regular al que se le da un identificador designado. Para definir el grupo con nombre se utiliza `(?P<nombre>` y `)` Por ejemplo, la siguiente expresión regular incluye un grupo con nombre con el identificador denominado `digits`:

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

Cuando se utiliza el método `exec()`, se añade un grupo con nombre coincidente como una propiedad del conjunto `result`:

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

Otro ejemplo, en el que se utilizan dos grupos con nombre, con los identificadores `name` y `dom`:

```
var emailPattern:RegExp =
    /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

Nota: *los grupos con nombre no forman parte de la especificación del lenguaje ECMAScript. Son una característica añadida de ActionScript 3.0.*

Indicadores y propiedades

En la tabla siguiente se muestran los cinco indicadores que se pueden establecer para expresiones regulares. Se puede acceder a cada indicador como una propiedad del objeto de expresión regular.

Indicador	Propiedad	Descripción
`g`	`global`	Detecta todas las coincidencias.
`i`	`ignoreCase`	Detecta sin distinguir mayúsculas de minúsculas. Se aplica a los caracteres `A`—`Z` y `a`—`z`, pero no a caracteres extendidos como `É` y `é`.
`m`	`multiline`	Con este indicador establecido, `$` y `^` pueden detectar el principio y el final de una línea, respectivamente.
`s`	`dotall`	Con este indicador establecido, `.` (punto) puede detectar el carácter de nueva línea (`\n`).
`x`	`extended`	Permite utilizar expresiones regulares extendidas. Estas expresiones permiten escribir espacios que se omitirán como parte del patrón. Esto facilita la lectura del código de una expresión regular.

Hay que tener en cuenta que estas propiedades son de sólo lectura. Se puede establecer los indicadores (`g`, `i`, `m`, `s`, `x`) al establecer una variable de expresión regular, de la manera siguiente:

```
var re:RegExp = /abc/gimsx;
```

Sin embargo, las propiedades con nombre no se pueden establecer directamente. Por ejemplo, el siguiente código genera un error:

```
var re:RegExp = /abc/;
re.global = true; // This generates an error.
```

De manera predeterminada, los indicadores no se establecen y las propiedades correspondientes se establecen en `false`, a menos que se especifiquen en la declaración de la expresión regular.

Además, las expresiones regulares tienen otras dos propiedades:

- La propiedad `lastIndex` especifica la posición del índice en la cadena que se debe utilizar para la siguiente llamada al método `exec()` o `test()` de una expresión regular.
- La propiedad `source` especifica la cadena que define la parte del patrón de la expresión regular.

El indicador g (global)

Si el indicador `g` (`global`) *no* se incluye, la expresión regular detectará una sola coincidencia. Por ejemplo, si no se incluye el indicador `g` en la expresión regular, el método `String.match()` devuelve una sola cadena coincidente:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

Si se establece el indicador `g`, el método `Sting.match()` devuelve varias coincidencias, como se indica a continuación:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// The same pattern, but this time the g flag IS set.
trace(str.match(pattern)); // output: she,shells,shore
```

El indicador i (ignoreCase)

De manera predeterminada, las expresiones regulares distinguen mayúsculas de minúsculas al detectar coincidencias. Si se establece el indicador `i` (`ignoreCase`), no se distinguirán mayúsculas de minúsculas. Por ejemplo, la `s` minúscula de la expresión regular no detecta la letra `S` mayúscula, el primer carácter de la cadena:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Not the first character
```

Sin embargo, con el indicador `i` establecido, la expresión regular detecta la letra `S` mayúscula:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

El indicador `i` no distingue mayúsculas de minúsculas únicamente para los caracteres `A–Z` y `a–z`, pero sí para caracteres extendidos como `É` y `é` .

El indicador m (multiline)

Si no se establece el indicador `m` (`multiline`), `^` detecta el principio de la cadena y `$` detecta el final de la cadena. Con `m` establecido, estos caracteres detectan el principio y el final de una línea, respectivamente. Considérese la siguiente cadena, que incluye un carácter de nueva línea:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Match a word at the beginning of the string.
```

Aunque se establezca el indicador `g` (`global`) en la expresión regular, el método `match()` detecta una sola subcadena, ya que hay una sola coincidencia con `^`, el principio de la cadena. El resultado es:

```
Test
```

A continuación se muestra el mismo código con el indicador `m` establecido:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Match a word at the beginning of lines.
```

Esta vez, el resultado incluye las palabras al principio de ambas líneas:

```
Test,Multiline
```

Hay que tener en cuenta que sólo el carácter `\n` indica el final de una línea. Los caracteres siguientes no:

- Carácter de retorno (`\r`)
- Carácter separador de línea Unicode (`\u2028`)

- Carácter separador de párrafo Unicode (`\u2029`)

El indicador s (dotall)

Si no se establece el indicador `s` (`dotall` o "dot all"), un punto (.) en un patrón de expresión regular no detectará un carácter de nueva línea (`\n`). Así, en el siguiente ejemplo no se detecta ninguna coincidencia:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

Sin embargo, si se establece el indicador `s`, el punto detecta el carácter de nueva línea:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

En este caso, la coincidencia es toda la subcadena entre las etiquetas `<p>`, incluido el carácter de nueva línea:

```
<p>Test
Multiline</p>
```

El indicador x (extended)

Las expresiones regulares pueden ser difíciles de leer, especialmente cuando incluyen muchos metasímbolos y metasecuencias. Por ejemplo:

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

Si se utiliza el indicador `x` (`extended`) en una expresión regular, se omitirán los espacios en blanco que se escriban en el patrón. Por ejemplo, la siguiente expresión regular es idéntica a la del ejemplo anterior:

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

Si se establece el indicador `x` y se desea detectar un carácter de espacio en blanco, se debe escribir una barra diagonal inversa inmediatamente antes del espacio en blanco. Por ejemplo, las dos expresiones normales siguientes son equivalentes:

```
/foo bar/
/foo \ bar/x
```

La propiedad lastIndex

La propiedad `lastIndex` especifica la posición de índice de la cadena en la que debe comenzar la siguiente búsqueda. Esta propiedad afecta a los métodos `exec()` y `test()` llamados en una expresión regular que tiene el indicador `g` establecido en `true`. Por ejemplo, considérese el fragmento de código siguiente:

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
    trace(pattern.lastIndex);
    result = pattern.exec(str);
}
```

La propiedad `lastIndex` está establecida en 0 de manera predeterminada (para iniciar las búsquedas al principio de la cadena). Después de cada detección, se establece en la posición de índice siguiente a la de la coincidencia. Por tanto, el resultado del código anterior es el siguiente:

```
0
5
11
18
25
36
44
```

Si el indicador `global` está establecido en `false`, los métodos `exec()` y `test()` no utilizan ni establecen la propiedad `lastIndex`.

Los métodos `match()`, `replace()` y `search()` de la clase String inician todas las búsquedas desde el principio de la cadena, independientemente del valor de la propiedad `lastIndex` de la expresión regular utilizada en la llamada al método. (Sin embargo, el método `match()` establece `lastIndex` en 0.)

Se puede establecer la propiedad `lastIndex` para ajustar la posición de inicio en la cadena para la detección de expresiones regulares.

La propiedad source

La propiedad `source` especifica la cadena que define la parte del patrón de una expresión regular. Por ejemplo:

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

Métodos para utilizar expresiones regulares con cadenas

La clase RegExp incluye dos métodos: `exec()` y `test()`.

Además de los métodos `exec()` y `test()` de la clase RegExp, la clase String incluye los siguientes métodos que permiten detectar expresiones regulares en cadenas: `match()`, `replace()`, `search()` y `splice()`.

El método test()

El método `test()` de la clase RegExp comprueba simplemente la cadena suministrada para ver si contiene una coincidencia para la expresión regular, como se indica en el siguiente ejemplo:

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

El método exec()

El método `exec()` de la clase RegExp comprueba la cadena suministrada para detectar una coincidencia con la expresión regular y devuelve un conjunto con lo siguiente:

- La subcadena coincidente
- La subcadena detecta grupos entre paréntesis en la expresión regular

El conjunto también incluye una propiedad `index`, que indica la posición del índice del inicio de la subcadena coincidente.

Por ejemplo, considérese el fragmento de código siguiente:

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //U.S phone number
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7-415-555-1212
```

El método `exec()` se utiliza varias veces para detectar varias subcadenas cuando se establece el indicador `g` (`global`) para la expresión regular:

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
    trace(result.index, "\t", pattern.lastIndex, "\t", result);
    result = pattern.exec(str);
}
//output:
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

Métodos de cadena que utilizan parámetros de tipo RegExp

Los siguientes métodos de la clase String toman como parámetros expresiones regulares: `match()`, `replace()`, `search()` y `split()`. Para más información sobre estos métodos, consulte "Búsqueda de patrones en cadenas y sustitución de subcadenas" en la página 150

Ejemplo: Analizador Wiki

Este ejemplo simple de conversión de texto de Wiki ilustra diversos usos de las expresiones regulares:

- Convertir líneas de texto que coinciden con un patrón de Wiki de origen con las cadenas HTML de salida apropiadas.
- Utilizar una expresión regular para convertir patrones de URL en etiquetas de hipervínculos HTML `<a>`.
- Uso de una expresión regular para convertir cadenas con valores monetarios en dólares de EE.UU. (como `"$9,95"`) en cadenas que contienen valores monetarios en euros (como `"8,24 €"`).

Para obtener los archivos de la aplicación para esta muestra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación WikiEditor se encuentran en la carpeta Samples/WikiEditor. La aplicación consta de los siguientes archivos:

Archivo	Descripción
WikiEditor.mxml o WikiEditor.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/regExpExamples/WikiParser.as	Una clase que incluye métodos que utilizan expresiones regulares para convertir patrones de texto de Wiki de origen en la salida HTML equivalente.
com/example/programmingas3/regExpExamples/URLParser.as	Una clase que incluye métodos que utilizan expresiones regulares para convertir cadenas de URL en etiquetas de hipervínculos HTML `<a>`.
com/example/programmingas3/regExpExamples/CurrencyConverter.as	Una clase que incluye métodos que utilizan expresiones regulares para convertir cadenas con valores monetarios en dólares de EE.UU. en cadenas en euros.

Definición de la clase WikiParser

La clase WikiParser incluye métodos que convierten texto Wiki de entrada en la salida HTML equivalente. Esta aplicación de conversión de Wiki no es muy sólida, pero ilustra algunos usos útiles de las expresiones regulares para la detección de patrones y la conversión de cadenas.

La función constructora, junto con el método `setWikiData()`, simplemente inicializa una cadena de ejemplo de texto de entrada de Wiki, de la manera siguiente:

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

Cuando el usuario hace clic en el botón Test de la aplicación de ejemplo, la aplicación invoca al método `parseWikiString()` del objeto WikiParser. Este método llama a otros métodos, que a su vez crean la cadena HTML resultante.

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

Cada uno de los métodos llamados, `parseBold()`, `parseItalic()`, `linesToParagraphs()` y `parseBullets()` utiliza el método `replace()` de la cadena para sustituir patrones coincidentes, definidos por una expresión regular, a fin de transformar el texto de entrada del Wiki en texto con formato HTML.

Convertir patrones en negrita y en cursiva

El método `parseBold()` busca un patrón de texto de Wiki en negrita (como `'''foo'''`) y lo transforma en su equivalente HTML (como `<b>foo</b>`), de la manera siguiente:

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

Hay que tener en cuenta que la parte `(.?*)` de la expresión regular detecta un número arbitrario de caracteres (`*`) entre los dos patrones delimitadores`'''`. El cuantificador `?` hace que no se detecte la mayor cantidad posible de caracteres, de forma que para una cadena como `'''aaa''' bbb '''ccc'''`, la primera cadena detectada será `'''aaa'''` y no la cadena completa (que empieza y termina con el patrón `'''`).

Los paréntesis de la expresión regular definen un grupo de captura y el método `replace()` hace referencia a este grupo mediante el código `$1` en la cadena de sustitución. El indicador `g` (`global`) de la expresión regular garantiza que el método `replace()` sustituye todas las coincidencias de la cadena (no sólo la primera).

El método `parseItalic()` funciona de forma similar al método `parseBold()`, con la diferencia de que busca dos apóstrofes (`''`), no tres, como delimitador del texto en cursiva:

```
private function parseItalic(input:String):String
{
    var pattern:RegExp = /''(.*?)''/g;
    return input.replace(pattern, "<i>$1</i>");
}
```

Conversión de patrones de viñetas

Como se indica en el siguiente ejemplo, el método `parseBullet()` busca el patrón de línea de viñeta del Wiki (como `* foo`) y la transforma en su equivalente HTML (como `<li>foo</li>`):

```
private function parseBullets(input:String):String
{
    var pattern:RegExp = /^\*(.*)/gm;
    return input.replace(pattern, "<li>$1</li>");
}
```

El símbolo `^` al principio de la expresión regular detecta el principio de una línea. El indicador `m` (`multiline`) de la expresión regular hace que la expresión regular detecte el símbolo `^` en cada principio de línea, no sólo al principio de la cadena.

El patrón `\*` detecta un asterisco (se utiliza la barra diagonal inversa para indicar un asterisco literal en lugar del cuantificador `*`).

Los paréntesis de la expresión regular definen un grupo de captura y el método `replace()` hace referencia a este grupo mediante el código `$1` en la cadena de sustitución. El indicador `g` (`global`) de la expresión regular garantiza que el método `replace()` sustituye todas las coincidencias de la cadena (no sólo la primera).

Conversión de patrones de párrafo de Wiki

El método `linesToParagraphs()` convierte cada línea de la cadena de entrada de Wiki en una etiqueta de párrafo HTML, `<p>`. Estas líneas del método eliminan las líneas vacías de la cadena de Wiki de entrada:

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

Los símbolos `^` y `$` hacen que la expresión regular detecte el principio y el final de una línea. El indicador `m` (`multiline`) de la expresión regular hace que la expresión regular detecte el símbolo ^ en cada principio de línea, no sólo al principio de la cadena.

El método `replace()` sustituye todas las subcadenas coincidentes (líneas vacías) con una cadena vacía (`""`). El indicador `g` (`global`) de la expresión regular garantiza que el método `replace()` sustituye todas las coincidencias de la cadena (no sólo la primera).

Conversión de direcciones URL en etiquetas HTML <a>

Cuando el usuario hace clic en el botón Test en la aplicación de ejemplo, si seleccionó la casilla de verificación `urlToATag`, la aplicación llama al método estático `URLParser.urlToATag()` para convertir cadenas de dirección URL de la cadena de Wiki de entrada en etiquetas HTML `<a>`.

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart, "ig");
var result:String = input.replace(urlPattern, "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

La función constructora `RegExp()` se utiliza para crear una expresión regular (`urlPattern`) a partir de varios constituyentes. Estos constituyentes son cadenas que definen partes del patrón de la expresión regular.

La primera parte del patrón de la expresión regular, definida por la cadena `protocol`, define un protocolo de URL: `http://` o `ftp://`. Los paréntesis definen un grupo que no captura, indicado por el símbolo `?`. Esto significa que los paréntesis se utilizan simplemente para definir un grupo para el patrón de alternancia `|`; el grupo no detectará códigos de elementos detectados previamente (`$1`, `$2`, `$3`) en la cadena de sustitución del método `replace()`.

Los otros elementos constituyentes de la expresión regular utilizan grupos de captura (indicados mediante paréntesis en el patrón), que se utilizan en los códigos de referencia a elementos detectados previamente (`$1`, `$2`, `$3`) en la cadena de sustitución del método `replace()`.

La parte del patrón definida por la cadena `urlPart` detecta *al menos* uno de los siguientes caracteres: `a-z`, `0-9`, `_` o `-`. El cuantificador `+` indica que debe detectarse al menos un carácter. `\.` indica un carácter punto (`.`) requerido. Y el resto detecta otra cadena que conste al menos de uno de los siguientes caracteres: `a-z`, `0-9`, `_` o `-`.

La parte del patrón definida por la cadena `optionalUrlPart` detecta *cero o más* de los caracteres siguientes: un punto (`.` seguido de cualquier número de caracteres alfanuméricos (incluidos `_` y `-`. El cuantificador `*` indica que deben detectarse cero o más caracteres.

La llamada al método `replace()` utiliza la expresión regular y crea la cadena HTML de sustitución utilizando referencias a elementos detectados previamente.

A continuación, el método `urlToATag()` llama al método `emailToATag()`, que utiliza técnicas similares para sustituir patrones de correo electrónico por cadenas de hipervínculos HTML `<a>`. Las expresiones regulares utilizadas para detectar direcciones URL HTTP, FTP y de correo electrónico en este archivo son bastante sencillas (para los fines del ejemplo); hay expresiones regulares mucho más complicadas para detectar direcciones URL de forma correcta.

Conversión de cadenas con valores monetarios en dólares de EE.UU. en cadenas con valores monetarios en euros

Cuando el usuario hace clic en el botón Test de la aplicación de ejemplo, si activó la casilla de verificación `dollarToEuro`, la aplicación llama al método estático `CurrencyConverter.usdToEuro()` para convertir cadenas con valores monetarios en dólares de EE.UU. (como `"$9.95"`) en cadenas con valores monetarios en euros (como `"8.24€"`), de la manera siguiente:

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

La primera línea define un patrón sencillo para detectar cadenas con valores monetarios en dólares de EE.UU. Obsérvese que antes del carácter `$` se escribe una barra diagonal inversa de escape (`\`).

El método `replace()` utiliza la expresión regular como detector de patrones y llama a la función `usdStrToEuroStr()` para determinar la cadena de sustitución (un valor en euros).

Cuando se utiliza un nombre de función como segundo parámetro del método `replace()`, se pasan a la función llamada los parámetros siguientes:

- La parte coincidente de la cadena.
- Las coincidencias detectadas por grupos de paréntesis de captura. El número de argumentos pasados de esta forma varía en función del número de capturas de grupos entre paréntesis. Se puede determinar el número de capturas de grupos entre paréntesis comprobando `arguments.length - 3` dentro del código de la función.
- La posición de índice en la que comienza la coincidencia en la cadena.
- La cadena completa.

El método `usdStrToEuroStr()` convierte patrones de cadena con valores monetarios en dólares de EE.UU. en cadenas con valores monetarios en euros, de la manera siguiente:

```
private function usdToEuro(...args):String
{
    var usd:String = args[1];
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.828017;
    var euro:Number = Number(usd) * exchangeRate;
    trace(usd, Number(usd), euro);
    const euroSymbol:String = String.fromCharCode(8364); // €
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Hay que tener en cuenta que `args[1]` representa la captura de grupo entre paréntesis detectada por la expresión regular `usdPrice`. Ésta es la parte numérica de la cadena de dólares de EE.UU., es decir, la cantidad de dólares sin el símbolo `$`. El método aplica una conversión de tasa de cambio y devuelve la cadena resultante (con un símbolo € final, en lugar de un símbolo $ inicial).

Capítulo 11: Trabajo con XML

ActionScript 3.0 incluye un grupo de clases basadas en la especificación de ECMAScript for XML (E4X) (ECMA-357 edición 2). Estas clases incluyen funciones eficaces y fáciles de usar para trabajar con datos XML. E4X permite desarrollar código con datos XML mucho más rápido que con las técnicas programación anteriores. Otra ventaja adicional es que el código que se cree será más fácil de leer.

En este capítulo se describe la manera de utilizar E4X para procesar datos XML.

Fundamentos de la utilización de XML

Introducción a la utilización de XML

XML es una forma estándar de representar información estructurada que los ordenadores pueden procesar fácilmente y que es razonablemente fácil de escribir y comprender para los humanos. XML es una abreviatura de eXtensible Markup Language (Lenguaje extensible de marcado). El estándar XML está disponible en www.w3.org/XML/.

XML ofrece una forma estándar y cómoda de clasificar datos y facilitar su lectura, acceso y manipulación. Utiliza una estructura de árbol y una estructura de etiquetas similares a las de HTML. A continuación se muestra un ejemplo sencillo de datos XML:

```
<song>
    <title>What you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Los datos XML también pueden ser más complejos, con etiquetas anidadas dentro de otras etiquetas así como atributos y otros componentes estructurales. A continuación se muestra un ejemplo más complejo de datos XML:

```
<album>
    <title>Questions, unanswered</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <tracks>
        <song tracknumber="1" length="4:05">
            <title>What do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:31</lastplayed>
        </song>
        <song tracknumber="2" length="3:45">
            <title>Who do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:35</lastplayed>
        </song>
        <song tracknumber="3" length="5:14">
            <title>When do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:39</lastplayed>
        </song>
        <song tracknumber="4" length="4:19">
            <title>Do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:44</lastplayed>
        </song>
    </tracks>
</album>
```

Este documento XML contiene otras estructuras XML completas (como las etiquetas `song` con sus elementos secundarios). También muestra otras estructuras XML como atributos (`tracknumber` y `length` en las etiquetas `song`) y etiquetas que contienen otras etiquetas en lugar de contener datos (como la etiqueta `tracks`).

Introducción a XML

A continuación se ofrece una descripción breve de los aspectos más comunes de los datos XML para usuarios con poca o ninguna experiencia en la utilización de XML. Los datos XML se escriben en formato de texto simple, con una sintaxis específica para organizar la información en un formato estructurado. Generalmente, un conjunto individual de datos XML se denomina *documento XML*. En formato XML, los datos se organizan en *elementos* (que pueden ser elementos de datos individuales o contenedores para otros elementos) con una estructura jerárquica. Cada documento XML tiene un elemento individual como elemento de nivel superior o principal; dentro de este elemento raíz puede haber un solo elemento de información, aunque es más probable que haya otros elementos, que a su vez contienen otros elementos, etc. Por ejemplo, este documento XML contiene información sobre un álbum de música:

```
<song tracknumber="1" length="4:05">
    <title>What do you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <mood>Happy</mood>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Cada elemento se distingue mediante un conjunto de *etiquetas*, constituidas por el nombre del elemento entre corchetes angulares (signos menor que y mayor que). La etiqueta inicial, que indica el principio del elemento, tiene el nombre de elemento:

```
<title>
```

La etiqueta final, que marca el final del elemento, tiene una barra diagonal antes del nombre del elemento:

```
</title>
```

Si un elemento no contiene nada, puede escribirse como un elemento vacío (y se representa con una sola etiqueta). En XML, este elemento:

```
<lastplayed/>
```

es idéntico a este elemento:

```
<lastplayed></lastplayed>
```

Además del contenido del elemento entre las etiquetas inicial y final, un elemento también puede incluir otros valores, denominados *atributos*, que se definen en la etiqueta inicial del elemento. Por ejemplo, este elemento XML define un solo atributo denominado `length`, con valor `"4:19"` :

```
<song length="4:19"></song>
```

Cada elemento XML tiene contenido, que puede ser un valor individual, uno o más elementos XML, o nada (en el caso de un elemento vacío).

Más información sobre XML

Para más información sobre la utilización de XML, hay vario libros y recursos adicionales, incluidos estos sitios Web:

- Tutorial XML de W3Schools: http://w3schools.com/xml/
- XML.com: http://www.xml.com/
- Tutoriales, listas de discusión y otros elementos de XMLpitstop: http://xmlpitstop.com/

Clases de ActionScript para trabajar con XML

ActionScript 3.0 incluye varias clases que se utilizan para trabajar con información estructurada en formato XML. Las dos clases principales son:

- XML: representa un solo elemento XML, que puede ser un documento XML con varios elementos secundarios o un elemento con un solo valor en un documento.
- XMLList: representa un conjunto de elementos XML. El objeto XMLList se utiliza cuando hay varios elementos XML del mismo nivel (están en el mismo nivel y pertenecen al mismo elemento principal en la jerarquía del documento XML). Por ejemplo, una instancia de XMLList sería la manera más sencilla de trabajar con este conjunto de elementos XML (que se supone contenido en un documento XML):

```
<artist type="composer">Fred Wilson</artist>
<artist type="conductor">James Schmidt</artist>
<artist type="soloist">Susan Harriet Thurndon</artist>
```

Para usos más avanzados que requieran espacios de nombres XML, ActionScript también incluye las clases Namespace y QName. Para más información, consulte “Utilización de espacios de nombres XML” en la página 247.

Además de las clases incorporadas para trabajar con XML, ActionScript 3.0 también incluye varios operadores que proporcionan funcionalidad específica para acceder a datos XML y manipularlos. Este enfoque para trabajar con XML mediante estas clases y operadores se denomina ECMAScript for XML (E4X) y está definido en la especificación de ECMA-357 edición 2.

Tareas comunes con XML

Al trabajar con XML en ActionScript es posible que haya que realizar las siguientes tareas con frecuencia:

- Crear documentos XML (añadir elementos y valores)
- Acceder a elementos, valores y atributos XML

- Filtrar elementos XML (buscando en ellos)
- Recorrer un conjunto de elementos XML
- Convertir datos entre las clases XML y la clase String
- Trabajo con espacios de nombres XML
- Cargar archivos XML externos

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Elemento: elemento individual de un documento XML, identificado como una etiqueta inicial y una etiqueta final, y el contenido existente entre las etiquetas (incluidas estas). Los elementos XML pueden contener texto o elementos de otro tipo, o pueden estar vacíos.
- Elemento vacío: elemento XML que no contiene elementos secundarios. Los elementos vacíos se suelen escribir con una sola etiqueta (como `<elemento/>`).
- Documento: estructura XML individual. Un documento XML puede contener un número arbitrario de elementos (o puede constar únicamente de un elemento vacío); no obstante, debe tener un solo elemento de nivel superior que contenga a todos los demás elementos del documento.
- Nodo: nombre alternativo para elemento XML.
- Atributo: valor con nombre asociado con un elemento que se escribe en la etiqueta inicial del elemento con el formato `nombreAtributo="valor"`, en lugar de escribirse como un elemento secundario independiente anidado dentro del elemento.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Prácticamente todos los listados de código de este capítulo ya incluyen la llamada a la función `trace()` apropiada. Para probar los listados de código de este capítulo:

1. Cree un documento de Flash vacío.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Ejecute el programa seleccionando Control > Probar película.

   El resultado de la función `trace()` se ve en el panel Salida.

Ésta y otras técnicas para probar los listados de código de ejemplo se describen de forma detallada en "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

El enfoque E4X del procesamiento de XML

La especificación de ECMAScript for XML define un conjunto de clases y funcionalidad para trabajar con datos XML. Este conjunto de clases y funcionalidades se denomina *E4X*. ActionScript 3.0 incluye las siguientes clases E4X: XML, XMLList, QName y Namespace.

Los métodos, propiedades y operadores de las clases de E4X se han diseñado con los siguientes objetivos:

- Simplicidad: siempre que sea posible, E4X facilita la escritura y comprensión del código para trabajar con datos XML.

- Coherencia: los métodos y la lógica que subyacen a E4X son coherentes internamente y con otros componentes de ActionScript.
- Familiaridad: los datos XML se manipulan con operadores conocidos, como el operador punto (.).

Nota: *ActionScript 2.0 tenía una clase XML. En ActionScript 3.0 se ha cambiado su nombre a XMLDocument para que no entre en conflicto con la clase XML de ActionScript 3.0 que forma parte de E4X. Las clases antiguas (XMLDocument, XMLNode, XMLParser y XMLTag) se incluyen en el paquete flash.xml principalmente por compatibilidad con código antiguo. Las nuevas clases de E4X son clases principales; no es necesario importar un paquete para utilizarlas. En este capítulo no se describe en detalle el uso de las clases XML antiguas de ActionScript 2.0. Para más detalles sobre estas clases, consulte el paquete flash.xml package en la Referencia del lenguaje y componentes ActionScript 3.0.*

A continuación se muestra un ejemplo de manipulación de datos con E4X:

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

A menudo, la aplicación cargará datos XML desde un origen externo, como un servicio Web o un canal RSS. No obstante, por claridad, los ejemplos de este capítulo asignan datos XML como literales.

Como se muestra en el código siguiente, E4X incluye algunos operadores intuitivos, como el operador punto (.) y el operador de identificador de atributo (@), para acceder a propiedades y atributos en datos XML:

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

El método `appendChild()` se utiliza para asignar un nuevo nodo secundario a los datos XML, como se indica en el siguiente fragmento de código:

```
var newItem:XML =
    <item id="3">
        <menuName>medium cola</menuName>
        <price>1.25</price>
    </item>

myXML.appendChild(newItem);
```

Los operadores `@` y `.` se utilizan no sólo para leer datos, sino también para asignar datos, como se indica a continuación:

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

Se puede utilizar un bucle `for` para recorrer nodos de los datos XML, de la manera siguiente:

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
    var q:int = Number(property.@quantity);
    var p:Number = Number(property.price);
    var itemTotal:Number = q * p;
    total += itemTotal;
    trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

Objetos XML

Un objeto XML puede representar un elemento, atributo, comentario, instrucción de procesamiento o elemento de texto XML.

Los objetos XML pueden clasificarse como de *contenido simple* o de *contenido complejo*. Un objeto XML que tiene nodos secundarios se clasifica como objeto de contenido complejo. Se dice que un objeto XML tiene contenido simple si es cualquiera de los siguientes elementos: un atributo, un comentario, una instrucción de procesamiento o un nodo de texto.

Por ejemplo, el siguiente objeto XML contiene contenido complejo, incluidos un comentario y una instrucción de procesamiento:

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
    <order>
        <!--This is a comment. -->
        <?PROC_INSTR sample ?>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Como se muestra en el siguiente ejemplo, ahora se pueden utilizar los métodos `comments()` y `processingInstructions()` para crear nuevos objetos XML, un comentario y una instrucción de procesamiento:

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

Propiedades XML

La clase XML tiene cinco propiedades estáticas:

- Las propiedades `ignoreComments` e `ignoreProcessingInstructions` determinan si deben omitirse los comentarios o las instrucciones de procesamiento cuando se analice el objeto XML.
- La propiedad `ignoreWhitespace` determina si deben omitirse los caracteres de espacio en blanco en las etiquetas de elemento y las expresiones incorporadas que sólo estén separadas por caracteres de espacio en blanco.

- Las propiedades `prettyIndentyprettyPrinting` se utilizan para aplicar formato al texto devuelto por los métodos `toString()` y `toXMLString()` de la clase XML.

Para obtener información sobre estas propiedades, consulte la Referencia del lenguaje y componentes ActionScript 3.0.

Métodos XML

Los siguientes métodos permiten trabajar con la estructura jerárquica de los objetos XML:

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`
- `insertChildBefore()`
- `parent()`
- `prependChild()`

Los siguientes métodos permiten trabajar con atributos de objetos XML:

- `attribute()`
- `attributes()`

Los siguientes métodos permiten trabajar con propiedades de objetos XML:

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

Los siguientes métodos sirven para trabajar con nombres completos y espacios de nombres:

- `addNamespace()`
- `inScopeNamespaces()`
- `localName()`
- `name()`
- `espacio de nombres()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

Los siguientes métodos sirven para trabajar con (y determinar) tipos específicos de contenido XML:

- `comments()`

- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

Los siguientes métodos sirven para la conversión a cadenas y para aplicar formato a objetos XML:

- `defaultSettings()`
- `setSettings()`
- `configuración()`
- `normalize()`
- `toString()`
- `toXMLString()`

Hay algunos métodos adicionales:

- `contains()`
- `copy()`
- `valueOf()`
- `longitud()`

Para obtener más información sobre estos métodos, consulte Referencia del lenguaje y componentes ActionScript 3.0.

Objetos XMLList

Una instancia de XMLList representa una colección arbitraria de objetos XML. Puede contener documentos XML completos, fragmentos XML o los resultados de una consulta XML.

Los siguientes métodos permiten trabajar con la estructura jerárquica de los objetos XMLList:

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

Los siguientes métodos permiten trabajar con atributos de objetos XMLList:

- `attribute()`
- `attributes()`

Los siguientes métodos permiten trabajar con las propiedades de XMLList:

- `hasOwnProperty()`
- `propertyIsEnumerable()`

Los siguientes métodos sirven para trabajar con (y determinar) tipos específicos de contenido XML:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

Los siguientes métodos sirven para la conversión a cadenas y para aplicar formato al objeto XMLList:

- `normalize()`
- `toString()`
- `toXMLString()`

Hay algunos métodos adicionales:

- `contains()`
- `copy()`
- `longitud()`
- `valueOf()`

Para obtener más información sobre estos métodos, consulte Referencia del lenguaje y componentes ActionScript 3.0.

Para un objeto XMLList que contiene exactamente un elemento XML se pueden utilizar todas las propiedades y métodos de la clase XML, ya que un objeto XMLList con un elemento XML se trata igual que un objeto XML. Por ejemplo, en el código siguiente, como `doc.div` es un objeto XMLList que contiene un elemento, se puede utilizar el método `appendChild()` de la clase XML:

```
var doc:XML =
        <body>
            <div>
                <p>Hello</p>
            </div>
        </body>;
doc.div.appendChild(<p>World</p>);
```

Para obtener una lista de propiedades y métodos XML, consulte "Objetos XML" en la página 237.

Inicialización de variables XML

Se puede asignar un literal XML a un objeto XML de la manera siguiente:

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Como se indica en el siguiente fragmento de código, también se puede utilizar el constructor `new` para crear una instancia de un objeto XML de una cadena que contiene datos XML:

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                            + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

Si los datos XML de la cadena no están bien formados (por ejemplo, si falta una etiqueta final), aparecerá un error en tiempo de ejecución.

También se puede pasar datos por referencia (desde otras variables) a un objeto XML, como se indica en el siguiente ejemplo:

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = "5";
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
    // Output: <item id="5">Chicken</item>
```

Para cargar datos XML desde una dirección URL hay que utilizar la clase URLLoader, como se indica en el siguiente ejemplo:

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

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
    var loader:URLLoader = event.target as URLLoader;
    if (loader != null)
    {
        externalXML = new XML(loader.data);
        trace(externalXML.toXMLString());
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

Para leer datos XML desde una conexión de socket hay que utilizar la clase XMLSocket. Para obtener más información, consulte la clase XMLSocket en la Referencia del lenguaje y componentes ActionScript 3.0.

Construcción y transformación de objetos XML

Los métodos `prependChild()` y `appendChild()` permiten añadir una propiedad al principio o al final (respectivamente) de una lista de propiedades de un objeto XML, como se indica en el siguiente ejemplo:

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
    // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

Los métodos `insertChildBefore()` e `insertChildAfter()` permiten añadir una propiedad antes o después (respectivamente) de una propiedad especificada, como se indica a continuación:

```
var x:XML =
    <body>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

Como se indica en el siguiente ejemplo, también se pueden utilizar operadores de llave ({ y }) para pasar datos por referencia (desde otras variables) al construir objetos XML:

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
    var newnode:XML = new XML();
    newnode =
        <employee id={ids[i]}>
            <last>{names[i][0]}</last>
            <first>{names[i][1]}</first>
        </employee>;

    x = x.appendChild(newnode)
}
```

Se pueden asignar propiedades y atributos a un objeto XML utilizando el operador =, como se indica a continuación:

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

Esto establece el valor del objeto XML `x` en:

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

Se pueden utilizar los operadores + y += para concatenar objetos XMLList.

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

Esto establece el valor del objeto XMLList `xList` en:

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

Navegación de estructuras XML

Una de las eficaces características de XML es su capacidad de proporcionar datos complejos y anidados a través de una cadena lineal de caracteres de texto. Al cargar datos en un objeto XML, ActionScript analiza los datos y carga su estructura jerárquica en memoria (o envía un error en tiempo de ejecución si los datos XML no están bien formados).

Los operadores y métodos de los datos XML y objetos XMLList facilitan la navegación de la estructura de datos XML.

El operador punto (.) y el operador descriptor de acceso descendiente (..) permiten acceder a propiedades secundarias de un objeto XML. Considere el siguiente objeto XML:

```
var myXML:XML =
    <order>
        <book ISBN="0942407296">
            <title>Baking Extravagant Pastries with Kumquats</title>
            <author>
                <lastName>Contino</lastName>
                <firstName>Chuck</firstName>
            </author>
            <pageCount>238</pageCount>
        </book>
        <book ISBN="0865436401">
            <title>Emu Care and Breeding</title>
            <editor>
                <lastName>Case</lastName>
                <firstName>Justin</firstName>
            </editor>
            <pageCount>115</pageCount>
        </book>
    </order>
```

El objeto `myXML.book` es un objeto XMLList que contiene propiedades secundarias del objeto `myXML` denominado `book`. Son dos objetos XML, que coinciden con las dos propiedades `book` del objeto `myXML`.

El objeto `myXML..lastName` es un objeto XMLList que contiene todas las propiedades de descendientes denominadas `lastName`. Son dos objetos XML, que coinciden con las dos propiedades `lastName` del objeto `myXML`.

El objeto `myXML.book.editor.lastName` es un objeto XMLList que contiene los elementos secundarios que tengan el nombre `lastName` de los elementos secundarios denominados `editor` de los elementos secundarios llamados `book` del objeto `myXML`: en este caso, es un objeto XMLList que contiene sólo un objeto XML (la propiedad `lastName` con el valor "`Case`").

Acceso a nodos principales y secundarios

El método `parent()` devuelve el elemento principal de un objeto XML.

Se pueden utilizar los valores de índice ordinales de una lista secundaria para acceder a objetos secundarios específicos. Por ejemplo, considérese un objeto XML `myXML` que tiene dos propiedades secundarias denominadas `book`. Cada propiedad secundaria denominada `book` tiene un número de índice asociado:

```
myXML.book[0]
myXML.book[1]
```

Para acceder a un elemento terciario específico se pueden especificar números de índice para los nombres del elemento secundario y el elemento terciario:

```
myXML.book[0].title[0]
```

Sin embargo, si `x.book[0]` sólo tuviera un elemento secundario denominado `title`, se puede omitir la referencia al índice, como se muestra a continuación:

```
myXML.book[0].title
```

De forma similar, si sólo hay un elemento secundario book del objeto `x` y dicho objeto secundario tiene un solo objeto de título, se pueden omitir ambas referencias de índice, como se muestra a continuación:

```
myXML.book.title
```

Se puede utilizar el método `child()` para desplazarse por los elementos secundarios con nombres basados en una variable o expresión, como se indica en el siguiente ejemplo:

```
var myXML:XML =
        <order>
            <book>
                <title>Dictionary</title>
            </book>
        </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

Acceso a atributos

El símbolo `@` (el operador identificador de atributo) se utiliza para acceder a atributos de un objeto XML o XMLList, como se muestra en el código siguiente:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

Se puede utilizar el símbolo de comodín `*` con el símbolo `@` para acceder a todos los atributos de un objeto XML o XMLList, como se muestra en el código siguiente:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

Se puede utilizar el método `attribute()` o `attributes()` para acceder a un atributo específico o a todos los atributos de un objeto XML o XMLList, como se muestra en el código siguiente:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

También se puede utilizar la sintaxis siguiente para acceder a atributos, como se muestra en el siguiente ejemplo:

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

Todos ellos equivalen a `employee.@id`. Sin embargo, la sintaxis `employee.@id` es la preferida.

Filtrado por atributo o valor de elemento

Se pueden utilizar los operadores de paréntesis, `(` y `)`, para filtrar elementos con un nombre de elemento o un valor de atributo específicos. Considere el siguiente objeto XML:

```
var x:XML =
    <employeeList>
        <employee id="347">
            <lastName>Zmed</lastName>
            <firstName>Sue</firstName>
            <position>Data analyst</position>
        </employee>
        <employee id="348">
            <lastName>McGee</lastName>
            <firstName>Chuck</firstName>
            <position>Jr. data analyst</position>
        </employee>
    </employeeList>
```

Las siguientes expresiones son todas válidas:

- `x.employee.(lastName == "McGee")`; es el segundo nodo `employee`.
- `x.employee.(lastName == "McGee").firstName`; es la propiedad `firstName` del segundo nodo `employee`.
- `x.employee.(lastName == "McGee").@id`; es el valor del atributo `id` del segundo nodo `employee`.

- `x.employee.(@id == 347)`; es el primer nodo `employee`.
- `x.employee.(@id== 347).lastName`; es la propiedad `lastName` del primer nodo `employee`.
- `x.employee.(@id > 300)`; es un objeto XMLList con ambas propiedades `employee`.
- `x.employee.(position.toString().search("analyst") > -1)`; es un objeto XMLList con ambas propiedades `position`.

Si se intenta filtrar por atributos o elementos que no existen, Flash® Player y Adobe® AIR™ emitirán una excepción. Por ejemplo, la línea final del código siguiente genera un error porque no hay ningún atributo `id` en el segundo elemento `p`:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(@id == '123'));
```

De manera similar, la línea final del código siguiente genera un error porque no hay ningún atributo `b` en el segundo elemento `p`:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(b == 'Bob'));
```

Para evitar estos errores, se pueden identificar las propiedades que tienen los atributos o elementos coincidentes mediante los métodos `attribute()` y `elements()`, como se muestra en el código siguiente:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

También se puede utilizar el método `hasOwnProperty()`, como se muestra en el código siguiente:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

Utilización de las sentencias for..in y for each..in

ActionScript 3.0 incluye las sentencias `for..in` y `for each..in` para recorrer los objetos XMLList. Por ejemplo, considérese el objeto XML `myXML` y el objeto XMLList `myXML.item`. El objeto XMLList, `myXML.item`, consta de los dos nodos `item` del objeto XML.

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2' quantity='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>;
```

La sentencia `for..in` permite recorrer un conjunto de nombres de propiedad de un objeto XMLList:

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

La sentencia `for each..in` permite recorrer las propiedades del objeto XMLList:

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

Utilización de espacios de nombres XML

Los espacios de nombres de un objeto (o documento) XML identifican el tipo de datos que el objeto contiene. Por ejemplo, al enviar y entregar datos XML a un servicio Web que utiliza el protocolo de mensajería SOAP, se declara el espacio de nombres en la etiqueta inicial de los datos XML:

```
var message:XML =
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <soap:Body xmlns:w="http://www.test.com/weather/">
            <w:getWeatherResponse>
                <w:tempurature >78</w:tempurature>
            </w:getWeatherResponse>
        </soap:Body>
    </soap:Envelope>;
```

El espacio de nombres tiene un prefijo, `soap`, y un URI que define el espacio de nombres, `http://schemas.xmlsoap.org/soap/envelope/`.

ActionScript 3.0 incluye la clase Namespace para trabajar con espacios de nombres XML. Para el objeto XML del ejemplo anterior se puede utilizar la clase Namespace de la manera siguiente:

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

La clase XML incluye los siguientes métodos para trabajar con espacios de nombres: `addNamespace()`, `inScopeNamespaces()`, `localName()`, `name()`, `namespace()`, `namespaceDeclarations()`, `removeNamespace()`, `setLocalName()`, `setName()` y `setNamespace()`.

La directiva `default xml namespace` permite asignar un espacio de nombres predeterminado para objetos XML. Por ejemplo, en el fragmento de código siguiente, `x1` y `x2` tienen el mismo espacio de nombres predeterminado:

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

Conversión de tipo XML

Se pueden convertir objetos XML y XMLList a valores de cadena. De forma similar, se pueden convertir cadenas en objetos XML y XMLList. También se debe tener en cuenta que todos los valores de atributos, nombres y valores de texto XML son cadenas. En las secciones siguientes se tratan todas estas formas de conversión de tipo XML.

Conversión de objetos XML y XMLList en cadenas

Las clases XML y XMLList incluyen un método `toString()` y un método `toXMLString()`. El método `toXMLString()` devuelve una cadena que incluye todas las etiquetas, los atributos, las declaraciones de espacios de nombres y el contenido del objeto XML. Para objetos XML con contenido complejo (elementos secundarios), el método `toString()` hace exactamente lo mismo que el método `toXMLString()`. Para objetos XML con contenido simple (los que contienen un solo elemento de texto), el método `toString()` devuelve únicamente el contenido de texto del elemento, como se indica en el siguiente ejemplo:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName.toXMLString());
    // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
    // burger
```

Si se utiliza el método `trace()` sin especificar `toString()` ni `toXMLString()`, los datos se convierten con el método `toString()` de manera predeterminada, como se muestra en el código siguiente:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName);
    // burger
```

Al utilizar el método `trace()` para depurar código, generalmente se deseará utilizar el método `toXMLString()` para que el método `trace()` devuelva datos más completos.

Conversión de cadenas a objetos XML

Se puede utilizar el constructor `new XML()` para crear un objeto XML de una cadena, de la manera siguiente:

```
var x:XML = new XML("<a>test</a>");
```

Si se intenta convertir una cadena en XML a partir de una cadena que representa datos XML no válidos o que no están bien formados, se emitirá un error en tiempo de ejecución, como se muestra a continuación:

```
var x:XML = new XML("<a>test"); // throws an error
```

Conversión de valores de atributos, nombres y valores de texto de tipo cadena

Todos los valores de atributos, nombres y valores de texto XML son del tipo de datos String y es posible que sea necesario convertirlos a otros tipos de datos. Por ejemplo, el código siguiente utiliza la función `Number()` para convertir valores de texto en números:

```
var myXML:XML =
                        <order>
                            <item>
                                <price>3.95</price>
                            </item>
                            <item>
                                <price>1.00</price>
                            </item>
                        </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
    myXML.total.children()[0] = Number(myXML.total.children()[0])
                                                + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.35;
```

Si este código no utilizara la función `Number()`, interpretaría el operador + como el operador de concatenación de cadenas y el método `trace()` en la última línea emitiría lo siguiente:

```
01.003.95
```

Lectura de documentos XML externos

Se puede utilizar la clase URLLoader para cargar datos XML desde una dirección URL. Para utilizar el código siguiente en las aplicaciones hay que sustituir el valor de `XML_URL` del ejemplo por una dirección URL válida:

```
var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener("complete", xmlLoaded);

function xmlLoaded(event:Event):void
{
    myXML = XML(myLoader.data);
    trace("Data loaded.");
}
```

También se puede utilizar la clase XMLSocket para configurar una conexión de socket XML asíncrona con un servidor. Para obtener más información, consulte Referencia del lenguaje y componentes ActionScript 3.0.

Ejemplo: Carga de datos de RSS desde Internet

La aplicación de ejemplo RSSViewer muestra diversas características del trabajo con XML en ActionScript, incluidas las siguientes:

- Utilización de métodos XML para recorrer datos XML en forma de canal RSS.
- Utilización de métodos XML para crear datos XML en formato HTML para utilizarlos en un campo de texto.

Se ha extendido el uso del formato RSS para sindicar noticias en formato XML. El aspecto de un archivo de datos RSS simple será similar al siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
    <title>Alaska - Weather</title>
    <link>http://www.nws.noaa.gov/alerts/ak.html</link>
    <description>Alaska - Watches, Warnings and Advisories</description>

    <item>
        <title>
            Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
        </title>
        <link>
            http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
</item>
    <item>
        <title>
            Short Term Forecast - Haines Borough (Alaska)
        </title>
            <link>
            http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
    </item>
</channel>
</rss>
```

La aplicación SimpleRSS lee datos RSS de Internet, analiza dichos datos en busca de titulares (títulos), vínculos y descripciones, y devuelve esos datos. La clase SimpleRSSUI proporciona la interfaz de usuario y llama a la clase SimpleRSS, que lleva a cabo todo el procesamiento de los datos XML.

Para obtener los archivos de la aplicación para esta muestra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación RSSViewer se encuentran en la carpeta Samples/RSSViewer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
RSSViewer.mxml o RSSViewer.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/rssViewer/RSSParser.as	Una clase que contiene métodos que utilizan E4X para atravesar datos RSS (XML) y generan una representación HTML correspondiente.
RSSData/ak.rss	Un archivo RSS de ejemplo. La aplicación está configurada para leer datos RSS de Internet, en un canal RSS de Flex alojado por Adobe. No obstante, se puede modificar fácilmente para que lea datos RSS de este documento, que utiliza un esquema ligeramente distinto del que utiliza el canal RSS de Flex.

Lectura y análisis de datos XML

La clase RSSParser incluye un método `xmlLoaded()` que convierte los datos RSS de entrada, almacenados en la variable `rssXML`, en una cadena que contiene la salida en formato HTML, `rssOutput`.

Casi al principio del método, el código establece el espacio de nombres XML predeterminado si el origen de datos RSS incluye un espacio de nombres predeterminado:

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

Las líneas siguientes recorren el contenido del origen de datos XML, examinando cada propiedad descendiente denominada `item`:

```
for each (var item:XML in rssXML..item)
{
    var itemTitle:String = item.title.toString();
    var itemDescription:String = item.description.toString();
    var itemLink:String = item.link.toString();
    outXML += buildItemHTML(itemTitle,
                            itemDescription,
                            itemLink);
}
```

Las tres primeras líneas son simplemente un conjunto de variables de cadena para representar las propiedades de título, descripción y vínculo de la propiedad `item` de los datos XML. A continuación, la siguiente línea llama al método `buildItemHTML()` para obtener datos HTML en forma de objeto XMLList, utilizando las tres nuevas variables de cadena como parámetros.

Construcción de datos XMLList

Los datos HTML (un objeto XMLList) tienen la siguiente forma:

```
<b>itemTitle</b>
<p>
    itemDescription
    <br />
    <a href="link">
        <font color="#008000">More...</font>
    </a>
</p>
```

Las primeras líneas del método borran el espacio de nombres XML predeterminado:

```
default xml namespace = new Namespace();
```

La directiva `default xml namespace` tiene ámbito de nivel de bloque de función. Esto significa que el ámbito de esta declaración es el método `buildItemHTML()`.

Las líneas siguientes crean el objeto XMLList basándose en los argumentos de cadena pasados a la función:

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
        // <font color="#008000"></font></a>
        // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

Este objeto XMLList representa una cadena datos adecuada para un campo de texto HTML de ActionScript.

El método `xmlLoaded()` utiliza el valor devuelto por el método `buildItemHTML()` y lo convierte en una cadena:

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

Extracción del título del canal RSS y envío de un evento personalizado

El método `xmlLoaded()` establece una variable de cadena `rssTitle` a partir de la información de los datos XML RSS de origen:

```
rssTitle = rssXML.channel.title.toString();
```

Por último, el método `xmlLoaded()` genera un evento, que notifica a la aplicación que los datos ha sido analizados y están disponibles:

```
            dataWritten = new Event("dataWritten", true);
```

Capítulo 12: Gestión de eventos

Un sistema de gestión de eventos permite a los programadores responder a la entrada del usuario y a los eventos del sistema de forma cómoda. El modelo de eventos de ActionScript 3.0 no sólo resulta cómodo, sino que también cumple los estándares y está perfectamente integrado en la lista de visualización de Adobe® Flash® Player y Adobe® AIR™. El nuevo modelo de eventos está basado en la especificación de eventos DOM (modelo de objetos de documento) de nivel 3, una arquitectura de gestión de eventos estándar, y constituye una herramienta de gestión de eventos intuitiva y de grandes prestaciones para los programadores de ActionScript.

Este capítulo se divide en cinco secciones. En las dos primeras se ofrece información general acerca de la gestión de eventos en ActionScript, Las últimas tres secciones describen los conceptos principales del modelo de eventos: flujo del evento, objeto de evento y detectores de eventos. El sistema de gestión de eventos de ActionScript 3.0 interactúa estrechamente con la lista de visualización, por lo que en este capítulo se da por hecho que se conoce el funcionamiento básico de dicha lista. Para obtener más información, consulte “Programación de la visualización” en la página 278.

Fundamentos de la gestión de eventos

Introducción a la gestión de eventos

Los eventos se pueden considerar como sucesos de cualquier tipo en el archivo SWF que resultan de interés para el programador. Por ejemplo, la mayor parte de los archivos SWF permiten algún tipo de interacción con el usuario, ya sea algo tan sencillo como responder a un clic del ratón o algo mucho más complejo, como aceptar y procesar datos escritos en un formulario. Toda interacción de este tipo entre el usuario y el archivo SWF se considera un evento. Los eventos también pueden producirse sin interacción directa con el usuario, como cuando se terminan de cargar datos desde un servidor o se activa una cámara conectada.

En ActionScript 3.0, cada evento se representa mediante un objeto de evento, que es una instancia de la clase Event o de alguna de sus subclases. Los objetos de evento no sólo almacenan información sobre un evento específico, sino que también contienen métodos que facilitan la manipulación de los objetos de evento. Por ejemplo, cuando Flash Player o AIR detectan un clic de ratón, crean un objeto de evento (una instancia de la clase MouseEvent) para representar ese evento de clic de ratón en concreto.

Tras crear un objeto de evento, Flash Player o AIR lo *distribuyen*, lo que significa que el objeto de evento se transmite al objeto que es el destino del evento. El objeto que actúa como destino del objeto de evento distribuido se denomina *destino del evento*. Por ejemplo, cuando se activa una cámara conectada, Flash Player distribuye un objeto de evento directamente al destino del evento que, en este caso, es el objeto que representa la cámara. No obstante, si el destino del evento está en la lista de visualización, el objeto de evento se hace pasar por la jerarquía de la lista de visualización hasta alcanzar el destino del evento. En algunos casos, el objeto de evento se "propaga" de vuelta por la jerarquía de la lista de visualización usando la misma ruta. Este recorrido por la jerarquía de la lista de visualización se denomina *flujo del evento*.

Se pueden usar detectores de eventos en el código para detectar los objetos de evento. Los *detectores de eventos* son las funciones o métodos que se escriben para responder a los distintos eventos. Para garantizar que el programa responde a los eventos, es necesario añadir detectores de eventos al destino del evento o a cualquier objeto de la lista de visualización que forme parte del flujo del evento de un objeto de evento.

Cuando se escribe código para un detector de eventos, se suele seguir esta estructura básica (los elementos en negrita son marcadores de posición que se sustituyen en cada caso específico):

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse);
```

Este código realiza dos acciones. En primer lugar, define una función, que es la forma de especificar las operaciones que se llevarán a cabo como respuesta al evento. A continuación, llama al método `addEventListener()` del objeto de origen, básicamente "suscribiendo" la función al evento especificado de modo que se lleven a cabo las acciones de la función cuando ocurra el evento. Cuando el evento se produce finalmente, el destino de evento comprueba la lista de todas las funciones y métodos registrados como detectores de eventos. A continuación los llama de uno en uno, pasando el objeto de evento como parámetro.

Es necesario modificar cuatro elementos del código para crear un detector de eventos personalizado. En primer lugar se debe cambiar el nombre de la función por el nombre que se desee usar (es necesario realizar este cambio en dos lugares, donde en el código aparece **eventResponse**). Seguidamente, hay que especificar el nombre de clase adecuado para el objeto de evento distribuido por el evento que se desea detectar (**EventType** en el código) y hay que indicar la constante apropiada para el evento específico (**EVENT_NAME** en el listado). En tercer lugar, es necesario llamar al método `addEventListener()` en el objeto que distribuirá el evento (**eventTarget** en este código). Opcionalmente, se puede cambiar el nombre de la variable utilizada como parámetro de la función (**eventObject** en el código).

Tareas comunes de la gestión de eventos

A continuación se enumeran diversas tareas comunes de la gestión de eventos que se describirán en este capítulo:

- Escribir código para responder a eventos
- Impedir que el código responda a eventos
- Trabajo con objetos de eventos
- Trabajo con el flujo del evento:
  - identificar la información del flujo del evento
  - Detener el flujo del evento
  - Impedir los comportamientos predeterminados
- Distribuir eventos desde las clases
- Creación de un tipo de evento personalizado

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Comportamiento predeterminado: algunos eventos incluyen un comportamiento que ocurre normalmente junto con el evento, denominado comportamiento predeterminado. Por ejemplo, cuando un usuario escribe texto en un campo de texto, se activa un evento de introducción de texto. El comportamiento predeterminado de ese evento es mostrar el carácter que se ha escrito en el campo de texto, si bien se puede sustituir ese comportamiento predeterminado (si, por alguna razón, no se desea que se muestre el carácter escrito).
- Distribuir: notificar a los detectores de eventos que se ha producido un evento.
- Evento: algo que le sucede a un objeto y que dicho objeto puede comunicar a otros.

- Flujo del evento: cuando los eventos se producen en un objeto de la lista de visualización (un objeto que se muestra en pantalla), todos los objetos que contienen ese objeto reciben la notificación del evento y, a su vez, notifican a sus detectores de eventos. El proceso comienza en el escenario y avanza a través de la lista de visualización hasta el objeto en el que se ha producido el evento para después volver de nuevo hasta el escenario. Este proceso se conoce como el flujo del evento.
- Objeto de evento: objeto que contiene información acerca de un evento en concreto y que se envía a todos los detectores cuando se distribuye un evento.
- Destino del evento: objeto que realmente distribuye el evento. Por ejemplo, si el usuario hace clic en un botón que se encuentra dentro de un objeto Sprite que, a su vez, está en el escenario, todos esos objetos distribuirán eventos, pero el destino del evento es el objeto en el que se produjo el evento; en este caso, el botón sobre el que se ha hecho clic.
- Detector: función u objeto que se ha registrado a sí mismo con un objeto para indicar que debe recibir una notificación cuando se produzca un evento específico.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Prácticamente todos los listados de código de este capítulo incluyen una llamada a una función `trace()` para probar los resultados del código. Para probar e listado de código de este capítulo:

1. Cree un documento vacío utilizando la herramienta de edición de Flash.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Ejecute el programa seleccionando Control > Probar película.

   El resultado de las funciones `trace()` del código se ve en el panel Salida.

Algunos de los ejemplos de código son más complejos y se han programado como una clase. Para probar estos ejemplos:

1. Cree un documento vacío utilizando la herramienta de edición de Flash y guárdelo en su equipo.
2. Cree un nuevo archivo de ActionScript y guárdelo en el mismo directorio que el documento creado en el paso 1. El nombre del archivo debe coincidir con el nombre de la clase del listado de código. Por ejemplo, si el listado de código define una clase denominada EventTest, use el nombre EventTest.as para guardar el archivo de ActionScript.
3. Copie el listado de código en el archivo de ActionScript y guarde el archivo.
4. En el documento, haga clic en una parte vacía del escenario o espacio de trabajo para activar el inspector de propiedades del documento.
5. En el inspector de propiedades, en el campo Clase de documento, escriba el nombre de la clase de ActionScript que copió del texto.
6. Ejecute el programa seleccionando Control > Probar película.

   Verá el resultado del ejemplo en el panel Salida.

En el capítulo "Prueba de los listados de código de ejemplo del capítulo" en la página 36 se explican de forma más detallada las técnicas para la comprobación de listados de código.

Diferencias entre la gestión de eventos en ActionScript 3.0 y en las versiones anteriores

La diferencia principal entre la gestión de eventos en ActionScript 3.0 y en las versiones anteriores radica en que en ActionScript 3.0 hay un único sistema para gestionar eventos, mientras que en las versiones anteriores de ActionScript existían varios. Esta sección comienza con una descripción general de la forma en la que funcionaba la gestión de eventos en las versiones anteriores de ActionScript y pasa luego a examinar el modo en que ha cambiado en ActionScript 3.0.

Gestión de eventos en versiones anteriores de ActionScript

Las versiones de ActionScript anteriores a la 3.0 ofrecen diversas formas de gestionar los eventos:

- Controladores de eventos `on()` que se pueden colocar directamente en instancias de Button y MovieClip.
- Controladores `onClipEvent()` que se pueden colocar directamente en instancias de MovieClip.
- Propiedades de funciones callback, como `XML.onload` y `Camera.onActivity`.
- Detectores de eventos que pueden registrarse con el método `addListener()`.
- La clase UIEventDispatcher que implementaba parcialmente el modelo de eventos DOM.

Cada uno de estos mecanismos presenta sus propias ventajas y limitaciones. Los controladores `on()` y `onClipEvent()` son fáciles de usar, pero dificultan el mantenimiento posterior de los proyectos, ya que el código colocado directamente en los botones y clips de películas puede ser difícil de encontrar. Las funciones callback también son sencillas de implementar, pero sólo permiten una función callback por evento. La implementación de los detectores de eventos es más compleja, ya que no sólo es necesario crear un objeto y una función de detector, sino también registrar el detector en el objeto que genera el evento. No obstante, este trabajo adicional permite crear varios objetos detectores y registrarlos todos para el mismo evento.

El desarrollo de componentes para ActionScript 2.0 dio lugar a un nuevo modelo de eventos. Este nuevo modelo, representado por la clase UIEventDispatcher, se basaba en un subconjunto de la especificación de eventos DOM, de modo que, para los desarrolladores que conozcan la gestión de eventos de componentes, la transición al nuevo modelo de eventos de ActionScript 3.0 resultará relativamente sencilla.

Por desgracia, la sintaxis utilizada en los distintos modelos de eventos es igual en algunos casos y diferente en otros. Por ejemplo, en ActionScript 2.0, algunas propiedades, como `TextField.onChanged`, se pueden usar como función callback o como detector de eventos. Sin embargo, la sintaxis para registrar objetos detectores varía dependiendo de si se utiliza una de las seis clases que admiten detectores en la clase UIEventDispatcher. Para las clases Key, Mouse, MovieClipLoader, Selection, Stage y TextField se debe usar el método `addListener()`, mientras que para la gestión de eventos de componentes es necesario utilizar un método llamado `addEventListener()`.

Otra complicación introducida por los distintos modelos de gestión de eventos es que el ámbito de la función de controlador de eventos varía ampliamente dependiendo del mecanismo usado. Dicho de otro modo, el significado de la palabra clave `this` varía entre los distintos sistemas de gestión de eventos.

Gestión de eventos en ActionScript 3.0

ActionScript 3.0 presenta un único modelo de gestión de eventos que sustituye a todos los mecanismos que existían en las versiones anteriores del lenguaje. El nuevo modelo de eventos se basa en la especificación de eventos DOM (modelo de objetos de documento) de nivel 3. Si bien el formato de archivo SWF no cumple específicamente con el estándar DOM, existen suficientes similitudes entre la lista de visualización y la estructura de DOM como para posibilitar la implementación del modelo de eventos DOM. Los objetos de la lista de visualización son análogos a los nodos de la estructura jerárquica de DOM y los términos *objeto de la lista de visualización* y *nodo* se usan de forma indistinta en este texto.

La implementación de Flash Player y AIR del modelo de eventos DOM incluye un concepto denominado comportamientos predeterminados. Un *comportamiento predeterminado* es una acción que Flash Player o AIR ejecutan como consecuencia normal de determinados eventos.

Comportamientos predeterminados

Normalmente, los desarrolladores son los responsables de escribir el código que responderá a los eventos. No obstante, en algunos casos un comportamiento está asociado con tal frecuencia a un evento, que Flash Player o AIR ejecutan automáticamente ese comportamiento a no ser que el desarrollador incluya código para cancelarlo. Flash Player o AIR muestran comportamientos de este tipo de forma automática, por lo que reciben el nombre de comportamientos predeterminados.

Por ejemplo, cuando un usuario escribe texto en un objeto TextField, resulta tan frecuente esperar que el texto se muestre en el objeto TextField que ese comportamiento se incorpora en Flash Player y AIR. Si no se desea que se produzca este comportamiento predeterminado, es posible cancelarlo usando el nuevo sistema de gestión de eventos. Cuando se escribe texto en un objeto TextField, Flash Player o AIR crean una instancia de la clase TextEvent para representar esa entrada de usuario. Si no se desea que Flash Player o AIR muestren el texto en el objeto TextField, es necesario acceder a esa instancia específica de TextEvent y llamar al método `preventDefault()` de la instancia.

No todos los comportamientos predeterminados se pueden impedir. Por ejemplo, Flash Player y AIR generan un objeto MouseEvent cuando el usuario hace doble clic en una palabra de un objeto TextField. El comportamiento predeterminado, que no se puede impedir, es resaltar la palabra que hay bajo el cursor.

Existen muchos tipos de objetos de eventos que no tienen ningún comportamiento predeterminado asociado. Por ejemplo, Flash Player distribuye un objeto de evento connect cuando se establece una conexión de red, pero no hay ningún comportamiento predeterminado asociado a él. En la documentación de la API referente a la clase Event y sus subclases se enumeran todos los tipos de eventos, y se describen los comportamientos predeterminados asociados, además de indicar si dicho comportamiento se puede impedir.

Es importante comprender que los comportamientos predeterminados sólo están asociados con objetos de eventos distribuidos por Flash Player o AIR y que no existen para objetos de eventos distribuidos mediante programación a través de ActionScript. Por ejemplo, se pueden usar los métodos de la clase EventDispatcher para distribuir un objeto de evento de tipo `textInput`, pero ese objeto de evento no tendrá ningún comportamiento predeterminado asociado. Es decir, Flash Player y AIR no mostrarán un carácter en un objeto TextField como resultado de un evento `textInput` que se haya distribuido mediante programación.

Novedades de los detectores de eventos de ActionScript 3.0

Para los desarrolladores con experiencia en el uso del método `addListener()` de ActionScript 2.0, puede resultar útil señalar las diferencias entre el modelo de detectores de eventos de ActionScript 2.0 y el de ActionScript 3.0. En la siguiente lista se muestran algunas de las diferencias principales entre los dos modelos de eventos:

- Para añadir detectores de eventos en ActionScript 2.0, es necesario usar `addListener()` en algunos casos y `addEventListener()` en otros, mientras que en ActionScript 3.0 siempre se utiliza `addEventListener()`.

- En ActionScript 2.0 no existe el flujo del evento, lo que quiere decir que el método `addListener()` sólo se puede llamar en el objeto que difunde el evento, mientras que en ActionScript 3.0, el método `addEventListener()` se puede llamar en cualquier objeto que forme parte del flujo del evento.
- En ActionScript 2.0, los detectores de eventos pueden ser funciones, métodos u objetos, mientras que en ActionScript 3.0 sólo las funciones o los métodos pueden ser detectores de eventos.

El flujo del evento

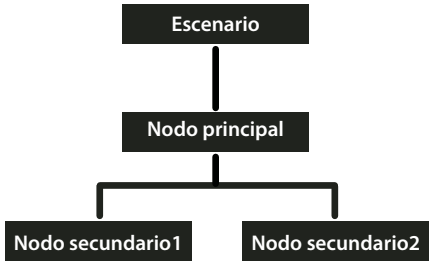
Flash Player o AIR distribuyen objetos de evento siempre que se produce un evento. Si el destino del evento no está en la lista de visualización, Flash Player o AIR distribuyen el objeto de evento directamente al destino del evento. Por ejemplo, Flash Player distribuye el objeto de evento progress directamente a un objeto URLStream. Sin embargo, si el destino del evento está en la lista de visualización, Flash Player distribuye el objeto de evento en la lista de visualización, de modo que recorre la lista hasta llegar al destino del evento.

El *flujo del evento* describe el modo en el que un objeto de evento se desplaza por la lista de visualización. La lista de visualización se organiza en una jerarquía que puede describirse como un árbol. En la parte superior de la jerarquía de la lista de visualización se encuentra el objeto Stage, que es un contenedor de objeto de visualización especial que actúa como raíz de la lista de visualización. El objeto Stage se representa mediante la clase flash.display.Stage y sólo es posible acceder a él a través de un objeto de visualización. Todos los objetos de visualización tienen una propiedad llamada `stage` que hace referencia al objeto Stage de esa aplicación.

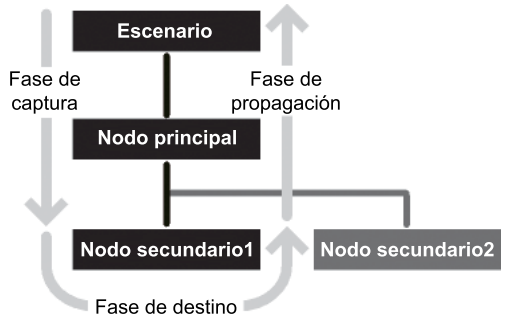
Cuando Flash Player o AIR distribuyen un objeto de evento para un evento relacionado con la lista de visualización, éste realiza un viaje de ida y vuelta desde el objeto Stage hasta el *nodo de destino*. La especificación de eventos DOM define el nodo de destino como el nodo que representa el destino del evento. Dicho de otro modo, el nodo de destino es el objeto de la lista de visualización en el que se ha producido el evento. Por ejemplo, si un usuario hace clic en un objeto de lista de visualización denominado `child1`, Flash Player o AIR distribuirán un objeto de evento usando `child1` como nodo de destino.

El flujo del evento se divide conceptualmente en tres partes. La primera parte se llama fase de captura y consta de todos los nodos desde el objeto Stage hasta el elemento principal del nodo de destino. La segunda parte se llama fase de destino y consiste solamente en el nodo de destino. La tercera parte se llama fase de propagación. La fase de propagación consta de los nodos encontrados en el viaje de vuelta desde el elemento principal del nodo de destino hasta el objeto Stage.

Los nombres de las fases cobran más sentido imaginando la lista de visualización como una jerarquía vertical con el objeto Stage en la parte superior, según se muestra en el diagrama siguiente:



Si un usuario hace clic en `Child1 Node`, Flash Player o AIR distribuyen un objeto de evento en el flujo del evento. Como muestra la imagen siguiente, el viaje del objeto empieza en `Stage`, desciende hasta `Parent Node`, luego avanza hasta `Child1 Node` y, finalmente, se propaga de vuelta hasta `Stage` cruzando de nuevo `Parent Node` en su vuelta a `Stage`.



En este ejemplo, la fase de captura consta de `Stage` y `Parent Node` durante el viaje descendente inicial. La fase de destino está compuesta por el tiempo empleado en `Child1 Node`. La fase de propagación consta de `Parent Node` y `Stage`, según se encuentran durante el viaje ascendente de vuelta hasta el nodo raíz.

El flujo del evento contribuye a lograr un sistema de gestión de eventos con mayores prestaciones que el que tenían anteriormente a su disposición los programadores de ActionScript. En las versiones anteriores de ActionScript, el flujo del evento no existe, de modo que los detectores de eventos sólo se pueden añadir al objeto que genera el evento. Por contra, en ActionScript 3.0, es posible añadir detectores de eventos no sólo a un nodo de destino, sino también a cualquier nodo que pertenezca al flujo del evento.

La posibilidad de añadir detectores de eventos a lo largo del flujo del evento resulta útil cuando un componente de la interfaz de usuario consta de más de un objeto. Por ejemplo, un objeto de botón suele contener un objeto de texto que actúa como etiqueta del botón. Sin la capacidad de añadir un detector al flujo del evento, sería necesario añadir un detector tanto al objeto de botón como al objeto de texto para garantizar que se reciben las notificaciones de eventos de clic que se producen en cualquier punto del botón. No obstante, gracias al flujo del evento es posible colocar un único detector de eventos en el objeto de botón para controlar los eventos de clic que se produzcan tanto en el objeto de texto como en las áreas del objeto de botón que no estén cubiertas por el objeto de texto.

Sin embargo, no todos los objetos participan en las tres fases del flujo del evento. Algunos tipos de eventos, como `enterFrame` e `init`, se distribuyen directamente al nodo de destino y no participan ni en la fase de captura ni en la de propagación. Otros eventos pueden tener como destino objetos que no aparecen en la lista de visualización, como los eventos distribuidos a las instancias de la clase Socket. Estos objetos de evento también van directamente al objeto de destino sin participar en las fases de captura y propagación.

Para conocer el comportamiento de un tipo particular de evento, se puede consultar la documentación de la API o examinar las propiedades del objeto de evento. En la siguiente sección se explica cómo examinar las propiedades del objeto de evento.

Objetos de evento

Los objetos de evento tienen una doble finalidad en el nuevo sistema de gestión de eventos. En primer lugar, representan los eventos reales, almacenando para ello información acerca de eventos específicos en un conjunto de propiedades. En segundo lugar, contienen un conjunto de métodos que permiten manipular objetos de evento y alterar el comportamiento del sistema de gestión de eventos.

Para facilitar el acceso a estas propiedades y métodos, la API de Flash Player define una clase Event que constituye la clase base de todos los objetos de evento. La clase Event define un conjunto fundamental de propiedades y métodos que son comunes a todos los objetos de evento.

Esta sección comienza con una explicación de las propiedades de la clase Event, prosigue con una descripción de los métodos de la clase Event y finaliza con un análisis de las razones por las que existen subclases de la clase Event.

Aspectos básicos de las propiedades de la clase Event

La clase Event define una serie de propiedades y constantes de sólo lectura que proporcionan información importante acerca de un objeto de evento. Las siguientes son especialmente importantes:

- Los tipos de objetos de evento se representan mediante constantes y se almacenan en la propiedad `Event.type`.
- La posibilidad de impedir el comportamiento predeterminado de un evento se representa mediante un valor booleano y se almacena en la propiedad `Event.cancelable`.
- La información del flujo del evento está contenida en las propiedades restantes.

Tipos de objetos de evento

Cada objeto de evento tiene un tipo de evento asociado. Los tipos de eventos se almacenan en la propiedad `Event.type` como valores de cadena. Resulta útil conocer el tipo de un objeto de evento, de manera que el código pueda distinguir objetos de distintos tipos. Por ejemplo, el siguiente código especifica que la función de detector `clickHandler()` debe responder a cualquier objeto de evento de clic del ratón que se pase a `myDisplayObject`:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

Existen unas dos docenas de tipos de eventos asociados a la clase Event, los cuales se representan con constantes de clase Event, algunas de las cuales se muestran en el siguiente extracto de la definición de la clase Event:

```
package flash.events
{
    public class Event
    {
        // class constants
        public static const ACTIVATE:String = "activate";
        public static const ADDED:String= "added";
        // remaining constants omitted for brevity
    }
}
```

Estas constantes proporcionan una forma sencilla de hacer referencia a tipos de eventos específicos. Es aconsejable utilizar estas constantes en lugar de las cadenas a las que representan. Si se escribe incorrectamente el nombre de una constante en el código, el compilador detectará el error, pero si se usan cadenas, un error tipográfico podría pasarse por alto durante la compilación y dar lugar a un comportamiento inesperado difícil de depurar. Por ejemplo, al añadir un detector de eventos, es preferible usar el siguiente código:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

en lugar de:

```
myDisplayObject.addEventListener("click", clickHandler);
```

Información sobre comportamientos predeterminados

El código puede comprobar si es posible impedir el comportamiento predeterminado de un objeto de evento en particular accediendo a la propiedad `cancelable`. La propiedad `cancelable` contiene un valor booleano que indica si es posible impedir un comportamiento predeterminado. Es posible impedir, o cancelar, el comportamiento predeterminado asociado a un reducido número de eventos usando el método `preventDefault()`. Para obtener más información, consulte Cancelación del comportamiento predeterminado de eventos en "Aspectos básicos de los métodos de la clase Event" en la página 263.

Información sobre el flujo del evento

Las demás propiedades de la clase Event contienen información importante acerca de un objeto de evento y su relación con el flujo del evento, según se describe en la siguiente lista:

- La propiedad `bubbles` contiene información acerca de las partes del flujo del evento en el que participa el objeto de evento.
- La propiedad `eventPhase` indica la fase en curso del flujo del evento.
- La propiedad `target` almacena una referencia al destino del evento.
- La propiedad `currentTarget` almacena una referencia al objeto de la lista de visualización que está procesando en ese momento el objeto de evento.

La propiedad bubbles

Se dice que un evento se propaga ("bubbles", en inglés) si su objeto de evento participa en la fase de propagación del flujo del evento, lo que quiere decir que dicho objeto regresa desde el nodo de destino, a través de sus ascendientes, hasta alcanzar el objeto Stage. La propiedad `Event.bubbles` almacena un valor booleano que indica si el objeto de evento participa en la fase de propagación. Dado que todos los eventos que se propagan también participan en las fases de captura y destino, cualquier evento que se propague participa en las tres fases del flujo del evento. Si el valor es `true`, el objeto de evento participa en las tres fases. Si es `false`, el objeto de evento no participa en la fase de propagación.

La propiedad eventPhase

Es posible determinar la fase del evento de un objeto de evento estudiando su propiedad `eventPhase`. La propiedad `eventPhase` contiene un valor entero sin signo que representa una de las tres fases del flujo del evento. La API de Flash Player define una clase EventPhase independiente que contiene tres constantes que se corresponden con los tres valores enteros sin signo, según se muestra en el siguiente extracto de código:

```
package flash.events
{
    public final class EventPhase
    {
        public static const CAPTURING_PHASE:uint = 1;
        public static const AT_TARGET:uint = 2;
        public static const BUBBLING_PHASE:uint= 3;
    }
}
```

Estas constantes corresponden a los tres valores válidos de la propiedad `eventPhase`. Es aconsejable usar estas constantes para hacer que el código sea más legible. Por ejemplo, para asegurarse de que una función denominada `miFunc()` sólo se llame si el destino del evento está en la fase de destino, es posible usar el siguiente código para probar dicha condición:

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
    myFunc();
}
```

La propiedad target

La propiedad `target` contiene una referencia al objeto que es el destino del evento. En algunas situaciones esto resulta evidente, como cuando se activa un micrófono, en cuyo caso el destino del objeto de evento es el objeto Microphone. Sin embargo, si el destino se encuentra en la lista de visualización, es necesario tener en cuenta la jerarquía de ésta. Por ejemplo, si un usuario hace clic con el ratón en un punto que incluye objetos solapados de la lista de visualización, Flash Player o AIR siempre seleccionan el objeto que se encuentra más lejos del objeto Stage como destino del evento.

En el caso de archivos SWF complejos, especialmente aquellos en los que los botones se decoran sistemáticamente con objetos secundarios menores, la propiedad `target` no puede usarse con demasiada frecuencia, ya que en muchas ocasiones señalará a los objetos secundarios de los botones en lugar de a los propios botones. En estas situaciones, lo habitual es añadir detectores de eventos al botón y usar la propiedad `currentTarget`, ya que ésta señala al botón, mientras que la propiedad `target` puede señalar a un elemento secundario del mismo.

La propiedad currentTarget

La propiedad `currentTarget` contiene una referencia al objeto que está procesando en ese momento al objeto de evento. Aunque puede parecer extraño no saber qué nodo está procesando en ese momento al objeto de evento que se está examinando, es necesario recordar que se puede añadir una función de detector a cualquier objeto de visualización en el flujo del evento de ese objeto de evento y que dicha función puede colocarse en cualquier lugar. Además, es posible añadir la misma función de detector a distintos objetos de visualización. A medida que el tamaño y la complejidad de un proyecto crecen, la propiedad `currentTarget` resulta cada vez más útil.

Aspectos básicos de los métodos de la clase Event

Existen tres categorías de métodos en la clase Event:

- Métodos de utilidad, que pueden crear copias de un objeto de evento o convertirlo en una cadena.
- Métodos de flujo del evento, que eliminan objetos de evento del flujo del evento.
- Métodos de comportamiento predeterminado, que impiden un comportamiento predeterminado o comprueban si se ha impedido.

Métodos de utilidad de la clase Event

Existen dos métodos de utilidad en la clase Event. El método `clone()`, que permite crear copias de un objeto de evento y el método `toString()`, que permite generar una representación de cadena de las propiedades de un objeto de evento junto con sus valores. El sistema de modelos de evento usa ambos métodos internamente, pero están a disposición de los desarrolladores para el uso general.

Los desarrolladores avanzados que deseen crear subclases de la clase Event deben sustituir e implementar versiones de ambos métodos de utilidad para asegurarse de que la subclase de eventos funcionará correctamente.

Detener el flujo del evento

Se puede llamar al método `Event.stopPropagation()` o `Event.stopImmediatePropagation()` para impedir que un objeto de evento siga moviéndose por el flujo del evento. Ambos métodos son casi idénticos y sólo se diferencian en que uno permite que se ejecuten los demás detectores de eventos del nodo en curso y el otro no:

- El método `Event.stopPropagation()` impide que el objeto de evento avance hasta el siguiente nodo, pero sólo después de permitir que se ejecuten todos los demás detectores de eventos del nodo en curso.
- El método `Event.stopImmediatePropagation()` también impide que el objeto de evento avance hasta el siguiente nodo, pero no permite que se ejecute ningún otro detector de eventos del nodo en curso.

Las llamadas a cualquiera de estos dos métodos no afectan a la aplicación del comportamiento predeterminado asociado a un evento. Es necesario usar los métodos de comportamiento predeterminado de la clase Event para impedir dicho comportamiento.

Cancelación del comportamiento predeterminado de eventos

Los dos métodos relacionados con la cancelación de comportamientos predeterminados son `preventDefault()` e `isDefaultPrevented()`. Para cancelar el comportamiento predeterminado asociado a un evento se usa el método `preventDefault()`, mientras que para comprobar si ya se ha llamado a `preventDefault()` en un objeto de evento, es necesario llamar al método `isDefaultPrevented()`, que devuelve el valor `true` si ya se ha llamado al método o `false` en caso contrario.

El método `preventDefault()` sólo funcionará si es posible cancelar el comportamiento predeterminado del evento. Se puede comprobar si esto es posible consultando la documentación de la API para ese tipo de evento o usando ActionScript para examinar la propiedad `cancelable` del objeto de evento.

La cancelación del comportamiento predeterminado no afecta al avance de un objeto de evento por el flujo del evento. Es necesario usar los métodos de flujo del evento de la clase Event para eliminar un objeto de evento del flujo del evento.

Subclases de la clase Event

Para multitud de eventos, el conjunto de propiedades comunes definido en la clase Event es suficiente. No obstante, otros eventos tienen características únicas que no es posible capturar mediante las propiedades de la clase Event. Para estos eventos, ActionScript 3.0 define varias subclases de la clase Event.

Cada subclase ofrece más propiedades y tipos de eventos que son únicas de esa categoría de eventos. Por ejemplo, los eventos relacionados con la entrada del ratón tienen algunas características únicas que no es posible capturar mediante las propiedades definidas en la clase Event. La clase MouseEvent amplía la clase Event añadiendo diez propiedades que contienen datos como la ubicación del evento de ratón y si se presionaron teclas específicas durante dicho evento.

Las subclases de Event también contienen constantes que representan los tipos de eventos asociados a la subclase. Por ejemplo, la clase MouseEvent define constantes para varios tipos de eventos de ratón e incluye los tipos de evento `click`, `doubleClick`, `mouseDown` y `mouseUp`.

Tal como se describe en la sección Métodos de utilidad de la clase Event en "Objetos de evento" en la página 260, al crear una subclase de Event es necesario sustituir los métodos `clone()` y `toString()` para ofrecer una funcionalidad específica de la subclase.

Detectores de eventos

Los detectores de eventos, también llamados controladores de eventos, son funciones que ejecutan Flash Player y AIR como respuesta a eventos específicos. El proceso de añadir un detector de eventos consta de dos pasos. En primer lugar se debe crear una función o método de clase para que Flash Player o AIR lo ejecuten como respuesta al evento. Esto a veces recibe el nombre de función de detector o función de controlador de eventos. En segundo lugar, es necesario usar el método `addEventListener()` para registrar la función de detector en el destino del evento o en cualquier objeto de la lista de visualización que se encuentre en el trayecto del flujo del evento adecuado.

Creación de funciones de detector

La creación de funciones de detector es un área en la que el modelo de eventos de ActionScript 3.0 difiere del modelo de eventos DOM. En el modelo de eventos DOM existe una distinción clara entre un detector de eventos y una función de detector: un detector de eventos es una instancia de una clase que implementa la interfaz EventListener, mientras que una función de detector es un método de esa clase denominada `handleEvent()`. En el modelo de eventos DOM, se debe registrar la instancia de la clase que contiene la función de detector en lugar de la función de detector en sí.

En el modelo de eventos de ActionScript 3.0 no hay distinción entre un detector de eventos y una función de detector. ActionScript 3.0 carece de una interfaz EventListener y es posible definir las funciones de detector fuera de una clase o como parte de ella. Además, no es necesario que las funciones de detector se denominen `handleEvent()`, sino que pueden usar cualquier identificador válido. En ActionScript 3.0 se registra el nombre de la función de detector real.

Función de detector definida fuera de una clase

El siguiente código crea un archivo SWF sencillo que muestra una forma cuadrada roja. Una función de detector denominada `clickHandler()`, que no forma parte de ninguna clase, detecta los eventos de clic del ratón sobre el cuadrado rojo.

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
}

function clickHandler(event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

Cuando el usuario interactúa con el archivo SWF resultante haciendo clic en el cuadrado, Flash Player o AIR generan la siguiente salida de traza:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Cabe destacar que el objeto de evento se pasa como argumento a `clickHandler()`. Esto permite a la función de detector examinar el objeto de evento. En este ejemplo se usa la propiedad `type` del objeto de evento para determinar si se trata de un evento de clic.

En el ejemplo también se comprueba el valor de la palabra clave `this`. En este caso, `this` representa el objeto global, lo cual es totalmente lógico, ya que la función se ha definido fuera de todo objeto o clase personalizada.

Función de detector definida como método de clase

El siguiente ejemplo es idéntico al anterior en el que se define la clase ClickExample, excepto en que la función `clickHandler()` se define como un método de la clase ChildSprite:

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
    private function clickHandler(event:MouseEvent):void
    {
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
    }
}
```

Cuando el usuario interactúa con el archivo SWF resultante haciendo clic en el cuadrado rojo, Flash Player o AIR generan la siguiente salida de traza:

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

La palabra clave `this` hace referencia a la instancia de ChildSprite denominada `child`. Se trata de un cambio de comportamiento respecto a ActionScript 2.0. Los usuarios que usaban componentes en ActionScript 2.0 sin duda recordarán que, al pasar un método de clase a `UIEventDispatcher.addEventListener()`, el ámbito del método estaba vinculado al componente que difundía el evento en lugar de a la clase en la que estaba definido el método de detector. Dicho de otro modo, si se usara esta técnica en ActionScript 2.0, la palabra clave `this` haría referencia al componente que realiza la difusión en lugar de a la instancia de ChildSprite.

Esto constituía un problema significativo para algunos programadores, ya que implicaba que no podían acceder a otros métodos y propiedades de la clase que contenía el método detector. Como solución, los programadores de ActionScript 2.0 podían usar la clase `mx.util.Delegate` para cambiar el ámbito del método detector. Esto ya no es necesario, ya que ActionScript 3.0 crea un método vinculado cuando se llama a `addEventListener()`. A consecuencia de ello, la palabra clave `this` hace referencia a la instancia de ChildSprite denominada `child` y el programador puede acceder a los demás métodos y propiedades de la clase ChildSprite.

Detector de eventos de uso no recomendado

Existe una tercera técnica, que no se recomienda, en la que se crea un objeto genérico con una propiedad que señala a una función de detector asignada dinámicamente. Se analiza aquí porque se solía utilizar en ActionScript 2.0, aunque no se debe emplear en ActionScript 3.0. Esta técnica no se recomienda, ya que la palabra clave `this` hará referencia al objeto global en lugar de al objeto detector.

El siguiente ejemplo es idéntico al anterior de la clase ClickExample, excepto en que la función de detector se define como parte de un objeto genérico denominado `myListenerObj`:

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
    }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
}
```

El resultado de la traza es el siguiente:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Cabría esperar que `this` hiciera referencia a `myListenerObj` y que la salida de traza fuese `[object Object]` pero, en vez de eso, hace referencia al objeto global. Al pasar el nombre de una propiedad dinámica como un argumento a `addEventListener()`, Flash Player o AIR no pueden crear un método vinculado. Esto se debe a que lo que se transmite como parámetro `listener` no es más que la dirección de memoria de la función de detector y Flash Player y AIR son incapaces de vincular esa dirección de memoria con la instancia de `myListenerObj` .

Administración de detectores de eventos

Es posible administrar las funciones de detector usando los métodos de la interfaz IEventDispatcher. La interfaz IEventDispatcher es la versión de ActionScript 3.0 de la interfaz EventTarget del modelo de eventos DOM. Si bien el nombre IEventDispatcher parece implicar que su objetivo principal es enviar (o distribuir) objetos de evento, en realidad los métodos de esta clase se usan con mucha más frecuencia para registrar detectores de eventos, comprobar su existencia y eliminarlos. La interfaz IEventDispatcher define cinco métodos, según se muestra en el siguiente código:

```
package flash.events
{
    public interface IEventDispatcher
    {
        function addEventListener(eventName:String,
                        listener:Object,
                        useCapture:Boolean=false,
                        priority:Integer=0,
                        useWeakReference:Boolean=false):Boolean;

        function removeEventListener(eventName:String,
                    listener:Object,
                    useCapture:Boolean=false):Boolean;

        function dispatchEvent(eventObject:Event):Boolean;

        function hasEventListener(eventName:String):Boolean;
        function willTrigger(eventName:String):Boolean;
    }
}
```

La API de Flash Player implementa la interfaz IEventDispatcher con la clase EventDispatcher, que sirve como clase base de todas las clases que pueden ser destinos de eventos o parte de un flujo de eventos. Por ejemplo, la clase DisplayObject hereda de la clase EventDispatcher. Esto quiere decir que cualquier objeto de la lista de visualización tiene acceso a los métodos de la interfaz IEventDispatcher.

Añadir detectores de eventos

El método `addEventListener()` es el elemento más ampliamente utilizado de la interfaz IEventDispatcher. Se utiliza para registrar las funciones de detector. Los dos parámetros necesarios son `type` y `listener`. Se puede usar el parámetro `type` para especificar el tipo de evento. El parámetro `listener`, por su parte, se emplea para especificar la función de detector que se ejecutará cuando se produzca el evento. El parámetro `listener` puede ser una referencia a una función o a un método de clase.

Nota: *no se deben usar paréntesis al especificar el parámetro* `listener`*. Por ejemplo, la función* `clickHandler()` *se especifica sin paréntesis en la siguiente llamada al método* `addEventListener()`*:*

Nota: `addEventListener(MouseEvent.CLICK, clickHandler)`.

El parámetro `useCapture` del método `addEventListener()` permite controlar la fase del flujo del evento en la que estará activa el detector. Si `useCapture` se establece en `true`, el detector estará activo durante la fase de captura del flujo del evento. Si `useCapture` se establece en `false`, el detector estará activo durante las fases de destino y de propagación del flujo del evento. Para detectar un evento durante todas las fases del flujo del evento se debe llamar a `addEventListener()` dos veces, una con `useCapture` establecido en `true` y luego otra con `useCapture` establecido en `false`.

El parámetro `priority` del método `addEventListener()` no es parte oficial del modelo de eventos DOM de nivel 3. Se incluye en ActionScript 3.0 para ofrecer una mayor flexibilidad a la hora de organizar los detectores de eventos. Cuando se llama a `addEventListener()`, es posible establecer la prioridad de ese detector de eventos pasando un valor entero como parámetro `priority`. El valor predeterminado es 0, pero se le pueden asignar valores enteros negativos o positivos. Cuanto mayor sea el número, antes se ejecutará el detector de eventos. Los detectores de eventos con la misma prioridad se ejecutan en el orden en que se añadieron, de modo que cuanto antes se añada, antes se ejecutará.

El parámetro `useWeakReference` permite especificar si la referencia a la función de detector es débil o normal. Si este parámetro se establece como `true` se pueden evitar situaciones en las que las funciones de detector permanecen en memoria incluso cuando ya no se necesitan. Flash Player y AIR usan una técnica denominada *eliminación de datos innecesarios* para borrar de la memoria objetos que han dejado de utilizarse. Se considera que un objeto ha dejado de usarse si no existe ninguna referencia a él. El recolector de datos innecesarios descarta las referencias débiles, de modo que una función de detector que sólo señala a una referencia débil puede ser eliminada, al considerarse como datos innecesarios.

Eliminación de detectores de eventos

Se puede usar el método `removeEventListener()` para eliminar los detectores de eventos que ya no sean necesarios. Es aconsejable eliminar todos los detectores que no vayan a usarse más. Los parámetros necesarios son `eventName` y `listener`, que son los mismos que para el método `addEventListener()`. Conviene recordar que se pueden detectar eventos durante todas las fases de eventos llamando a `addEventListener()` dos veces, una vez con `useCapture` establecido en `true` y luego otra establecido en `false`. Para eliminar los dos detectores de eventos sería necesario llamar a `removeEventListener()` dos veces, una con `useCapture` establecido en `true` y luego otra establecido en `false`.

Distribución de eventos

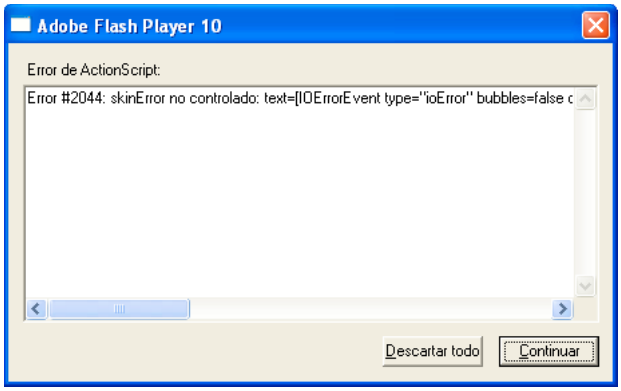
Los programadores expertos pueden usar el método `dispatchEvent()` para distribuir un objeto de evento personalizado en el flujo del evento. Este método sólo acepta un parámetro, consistente en una referencia a un objeto de evento, que debe ser una instancia de la clase Event o una subclase de ésta. Una vez distribuido, la propiedad `target` del objeto de evento se establece en el objeto en el que se llamó a `dispatchEvent()`.

Comprobación de la existencia de detectores de eventos

Los dos últimos métodos de la interfaz IEventDispatcher ofrecen información útil sobre la existencia de detectores de eventos. El método `hasEventListener()` devuelve `true` si se encuentra un detector de eventos para un tipo de evento específico en un objeto concreto de la lista de visualización. El método `willTrigger()` también devuelve `true` si se encuentra un detector para un objeto concreto de la lista de visualización, pero `willTrigger()` no sólo comprueba la existencia de detectores en el objeto de la lista de visualización, sino también en todos los ascendientes del objeto de la lista de visualización para todas las fases del flujo del evento.

Eventos de error sin detectores

Las excepciones, y no los eventos, son el principal mecanismo de gestión de errores en ActionScript 3.0, pero la gestión de excepciones no funciona en operaciones asíncronas, como la carga de archivos. Si se produce un error durante una de estas operaciones asíncronas, Flash Player y AIR distribuyen un objeto de evento de error. Si no se crea un detector para el evento de error, las versiones de depuración de Flash Player y AIR mostrarán un cuadro de diálogo con información acerca del error. Por ejemplo, la versión del depurador de Flash Player genera el siguiente cuadro de diálogo que describe el error cuando la aplicación intenta cargar un archivo desde una dirección URL no válida:



La mayor parte de los eventos de error se basan en la clase ErrorEvent y, por lo tanto, tienen una propiedad denominada `text` que se usa para almacenar el mensaje de error que muestra Flash Player o AIR. Las dos excepciones a esta regla son las clases StatusEvent y NetStatusEvent. Ambas clases tienen una propiedad `level` (`StatusEvent.level` y `NetStatusEvent.info.level`). Cuando el valor de la propiedad `level` es "`error`", estos tipos de evento se consideran eventos de error.

Un evento de error no hace que el archivo SWF deje de ejecutarse. Sólo se manifestará como un cuadro de diálogo en las versiones de depuración de los plugins de navegadores y de los reproductores autónomos, como un mensaje en el panel de salida del reproductor de edición y como una entrada en el archivo de registro de Adobe Flex Builder 3. No se manifiesta en las versiones oficiales de Flash Player o AIR.

Ejemplo: Reloj con alarma

El ejemplo del reloj con alarma consiste en un reloj que permite al usuario especificar una hora a la que sonará una alarma y se mostrará un mensaje. El ejemplo del reloj con alarma se basa en la aplicación SimpleClock de "Trabajo con fechas y horas" en la página 135 El ejemplo ilustra diversos aspectos de la utilización de eventos en ActionScript 3.0 como, por ejemplo:

- Detección y respuesta a un evento
- Notificación a los detectores de un evento
- Creación de un tipo de evento personalizado

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación del reloj con alarma se encuentran en la carpeta Samples/AlarmClock. La aplicación consta de los siguientes archivos:

Archivo	Descripción
AlarmClockApp.mxml o AlarmClockApp.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/clock/AlarmClock.as	Una clase que amplía la clase SimpleClock y añade la función de alarma al reloj.
com/example/programmingas3/clock/AlarmEvent.as	Una clase de eventos personalizada (una subclase de flash.events.Event) que actúa como el objeto de evento del evento `alarm` de la clase AlarmClock.
com/example/programmingas3/clock/AnalogClockFace.as	Dibuja una esfera de reloj redonda y las manecillas de hora, minutos y segundos, en función de la hora (descrita en el ejemplo SimpleClock).
com/example/programmingas3/clock/SimpleClock.as	Un componente de la interfaz de reloj con funciones sencillas de control de tiempo (descrito en el ejemplo de SimpleClock).

Información general sobre el reloj con alarma

Para la funcionalidad principal del reloj de este ejemplo, incluido el control del tiempo y la visualización de la esfera del reloj, se vuelve a utilizar el código de la aplicación SimpleClock, que se describe en "Ejemplo: Sencillo reloj analógico" en la página 140. La clase AlarmClock amplía la clase SimpleClock del ejemplo añadiendo la funcionalidad necesaria para un reloj con alarma, incluido el ajuste de la hora de la alarma y la notificación cuando suena la alarma.

Los eventos están diseñados para proporcionar notificaciones cuando ocurre algo. La clase AlarmClock expone el evento Alarm, sobre el que los demás objetos pueden realizar detecciones a fin de llevar a cabo las acciones deseadas. Además, la clase AlarmClock usa una instancia de la clase Timer para determinar cuándo hay que activar la alarma. Al igual que la clase AlarmClock, la clase Timer proporciona un evento para notificar a los demás objetos (una instancia de AlarmClock en este caso) cuándo ha transcurrido una determinada cantidad de tiempo. Tal y como ocurre con la mayoría de las aplicaciones de ActionScript, los eventos forman una parte importante de la funcionalidad de la aplicación del ejemplo del reloj con alarma.

Activación de la alarma

Según se ha mencionado con anterioridad, la única funcionalidad que la clase AlarmClock ofrece realmente está relacionada con la configuración y activación de la alarma. La clase integrada Timer (flash.utils.Timer) proporciona un mecanismo para que los desarrolladores definan código que se ejecutará tras un período de tiempo especificado. La clase AlarmClock utiliza una instancia de Timer para determinar cuándo activar la alarma.

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * The Timer that will be used for the alarm.
 */
public var alarmTimer:Timer;
...
/**
 * Instantiates a new AlarmClock of a given size.
 */
public override function initClock(faceSize:Number = 200):void
{
    super.initClock(faceSize);
    alarmTimer = new Timer(0, 1);
    alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

La instancia de Timer definida en la clase AlarmClock se denomina `alarmTimer`. El método `initClock()`, que lleva a cabo las operaciones de configuración necesarias para la instancia de AlarmClock, realiza dos operaciones con la variable `alarmTimer`. En primer lugar, se crea una instancia de la variable con parámetros que indican a la instancia Timer que debe esperar 0 milisegundos y activar su evento de temporizador sólo una vez. Tras crear la instancia de `alarmTimer`, el código llama al método `addEventListener()` de la variable para indicar que desea realizar una detección en el evento `timer` de esa variable. Las instancias de Timer funcionan distribuyendo su evento `timer` una vez transcurrida una cantidad de tiempo especificada. La clase AlarmClock necesitará saber cuándo se distribuye el evento `timer` para activar su propia alarma. Al llamar a `addEventListener()`, el código de AlarmClock se registra como detector con `alarmTimer`. Los dos parámetros indican que la clase AlarmClock desea detectar el evento `timer` (indicado por la constante `TimerEvent.TIMER`) y que, cuando éste se produzca, debe llamarse al método `onAlarm()` de la clase AlarmClock como respuesta al evento.

Para establecer la alarma, se llama al método `setAlarm()` de la clase AlarmClock del siguiente modo:

```
/**
 * Sets the time at which the alarm should go off.
 * @param hour The hour portion of the alarm time.
 * @param minutes The minutes portion of the alarm time.
 * @param message The message to display when the alarm goes off.
 * @return The time at which the alarm will go off.
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String = "Alarm!"):Date
{
    this.alarmMessage = message;
    var now:Date = new Date();
    // Create this time on today's date.
    alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

    // Determine if the specified time has already passed today.
    if (alarmTime <= now)
    {
        alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
    }

    // Stop the alarm timer if it's currently set.
    alarmTimer.reset();
    // Calculate how many milliseconds should pass before the alarm should
    // go off (the difference between the alarm time and now) and set that
    // value as the delay for the alarm timer.
    alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
    alarmTimer.start();

    return alarmTime;
}
```

Este método realiza varias tareas, entre ellas almacenar el mensaje de alarma y crear un objeto Date (`alarmTime`) que representa el instante de tiempo real en el que debe sonar la alarma. En las últimas líneas del método, el temporizador de la variable `alarmTimer` se define y se activa, lo que resulta de especial relevancia para este análisis. En primer lugar, se llama a su método `reset()`, deteniendo el temporizador y reiniciándolo en caso de que ya se estuviese ejecutando. A continuación, se resta la hora actual (representada por la variable `now`) del valor de la variable `alarmTime` para determinar cuántos milisegundos tienen que transcurrir antes de que suene la alarma. La clase Timer no activa su evento `timer` a una hora absoluta, de manera que es esta diferencia relativa de tiempo la que se asigna a la propiedad `delay` de `alarmTimer`. Finalmente, se llama al método `start()` para iniciar el temporizador.

Una vez que ha transcurrido la cantidad especificada de tiempo, `alarmTimer` distribuye el evento `timer`. Dado que la clase AlarmClock ha registrado su método `onAlarm()` como un detector de ese evento, cuando se produzca el evento `timer` se llamará a `onAlarm()`.

```
/**
 * Called when the timer event is dispatched.
 */
public function onAlarm(event:TimerEvent):void
{
    trace("Alarm!");
    var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
    this.dispatchEvent(alarm);
}
```

Un método que se registra como detector de eventos debe definirse con la firma adecuada (es decir, el conjunto de parámetros y el tipo de devolución del método). Para que un método pueda ser detector del evento `timer` de la clase Timer, debe definir un parámetro cuyo tipo de datos sea TimerEvent (flash.events.TimerEvent), una subclase de la clase Event. Cuando la instancia de Timer llama a sus detectores de eventos, pasa una instancia TimerEvent como objeto de evento.

Notificación de la alarma a otros

Al igual que la clase Timer, la clase AlarmClock proporciona un evento que permite que otro código reciba notificaciones cuando suena la alarma. Para que una clase pueda usar el marco de gestión de eventos incorporado en ActionScript, debe implementar la interfaz flash.events.IEventDispatcher. Normalmente esto se lleva a cabo ampliando la clase flash.events.EventDispatcher, que proporciona una implementación estándar de IEventDispatcher (o ampliando una de las subclases de EventDispatcher). Según se ha descrito anteriormente, la clase AlarmClock amplía la clase SimpleClock que, a su vez, amplía la clases Sprite, que (a través de una cadena de herencias) amplía la clase EventDispatcher. Todo esto quiere decir que la clase AlarmClock ya incorpora la funcionalidad adecuada para proporcionar sus propios eventos.

Se puede registrar otro código para que reciba notificaciones del evento `alarm` de la clase AlarmClock llamando al método `addEventListener()` que AlarmClock hereda de EventDispatcher. Cuando una instancia de AlarmClock está lista para notificar a otro código que su evento `alarm` se ha activado, lleva a cabo esta operación llamando al método `dispatchEvent()`, que también se hereda de EventDispatcher.

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

Estas líneas de código están tomadas del método `onAlarm()` de la clase AlarmClock (que se ha mostrado al completo anteriormente). Se llama al método `dispatchEvent()` de la instancia de AlarmClock que, a su vez, notifica a todos los detectores registrados que el evento `alarm` de la instancia de AlarmClock se ha activado. El parámetro que se pasa a `dispatchEvent()` es el objeto de evento que se pasará a los métodos detectores. En este caso es una instancia de la clase AlarmEvent, una subclase de Event creada específicamente para este ejemplo.

Creación de un evento de alarma personalizado

Todos los detectores de eventos reciben un parámetro de objeto de evento con información acerca del evento específico que se está activando. En muchos casos, el objeto de evento es una instancia de la clase Event. No obstante, en algunas ocasiones resulta útil proporcionar más información a los detectores de eventos. Según se ha explicado anteriormente en este capítulo, una forma habitual de lograr esto es definir una nueva clase (una subclase de la clase Event) y usar una instancia de esa clase como objeto de evento. En este ejemplo, se usa una instancia de AlarmEvent como objeto de evento cuando se distribuye el evento `alarm` de la clase AlarmClock. La clase AlarmEvent, que se muestra aquí, ofrece más información acerca del evento `alarm`, específicamente el mensaje de alarma:

```
import flash.events.Event;

/**
 * This custom Event class adds a message property to a basic Event.
 */
public class AlarmEvent extends Event
{
    /**
     * The name of the new AlarmEvent type.
     */
    public static const ALARM:String = "alarm";

    /**
     * A text message that can be passed to an event handler
     * with this event object.
     */
    public var message:String;

    /**
     *Constructor.
     *@param message The text to display when the alarm goes off.
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
        super(ALARM);
        this.message = message;
    }
    ...
}
```

La mejor forma de crear una clase de objetos de evento personalizados es definir una clase que amplíe la clase Event, según se muestra en el ejemplo anterior. Para complementar la funcionalidad heredada, la clase AlarmEvent define una propiedad `message` que contiene el texto del mensaje de alarma asociado al evento; el valor de `message` se pasa como un parámetro en el constructor de AlarmEvent. La clase AlarmEvent también define la constante `ALARM`, que se puede utilizar para hacer referencia al evento específico (`alarm`) al llamar al método `addEventListener()` de la clase AlarmClock.

Además de añadir funcionalidad personalizada, cada subclase de Event debe sustituir el método `clone()` heredado como parte del marco de gestión de eventos de ActionScript. Las subclases Event también pueden sustituir el método heredado `toString()` para incluir las propiedades del evento personalizado en el valor que se devuelve al llamar al método `toString()`.

```
    /**
     * Creates and returns a copy of the current instance.
     * @return A copy of the current instance.
     */
    public override function clone():Event
    {
        return new AlarmEvent(message);
    }

    /**
     * Returns a String containing all the properties of the current
     * instance.
     * @return A string representation of the current instance.
     */
    public override function toString():String
    {
        return formatToString("AlarmEvent", "type", "bubbles", "cancelable", "eventPhase",
"message");
    }
```

El método `clone()` sustituido necesita devolver una nueva instancia de la subclase personalizada de Event con todas las propiedades personalizadas definidas para coincidir con la instancia actual. En el método `toString()` sustituido, el método de utilidad `formatToString()` (heredado de Event) se utiliza para proporcionar una cadena con el nombre del tipo personalizado, además de los nombres y valores de todas sus propiedades.

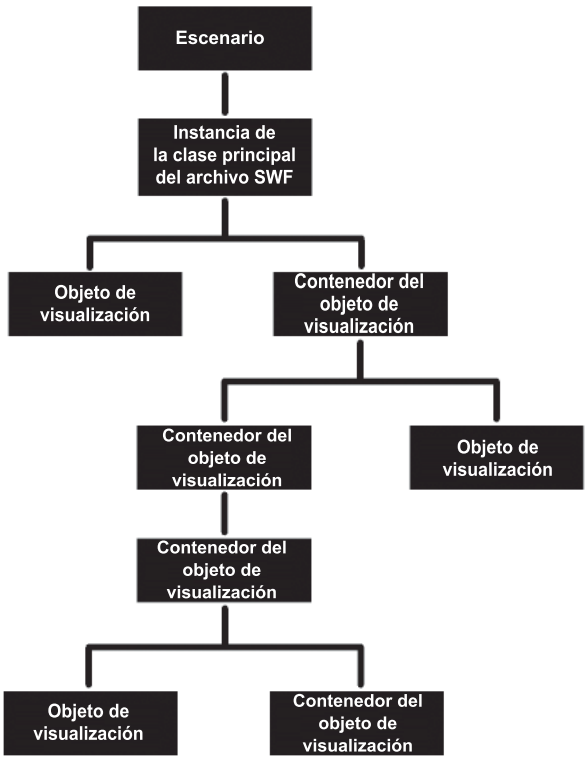
Capítulo 13: Programación de la visualización

La programación de la visualización en Adobe® ActionScript® 3.0 permite trabajar con elementos que aparecen en el escenario de Adobe® Flash® Player o Adobe® AIR™. En este capítulo se describen los conceptos básicos para trabajar con elementos en pantalla. Se ofrecerán detalles acerca de la organización de elementos visuales mediante programación. El usuario también aprenderá a crear sus propias clases personalizadas para objetos de visualización.

Fundamentos de la programación de la visualización

Introducción a la programación de la visualización

Cada aplicación creada con ActionScript 3.0 tiene una jerarquía de objetos visualizados, conocida como *lista de visualización*, tal y como se muestra a continuación. La lista de visualización contiene todos los elementos visibles en la aplicación.



Tal y como muestra la ilustración, los elementos en pantalla se ubican en uno o varios de los siguientes grupos:

- Objeto Stage

  El objeto Stage es el contenedor base de los objetos de visualización. Cada aplicación tiene un objeto Stage, que contiene todos los objetos de visualización en pantalla. El objeto Stage es el contenedor de nivel superior y se encuentra arriba del todo en la jerarquía de la lista de visualización:

  Cada archivo SWF tiene una clase de ActionScript asociada, conocida como *la clase principal del archivo SWF*. Cuando se abre un archivo SWF en Flash Player o en Adobe AIR, Flash Player o AIR llama a la función constructora de dicha clase y la instancia que se crea (que es siempre un tipo de objeto de visualización) se añade como elemento secundario del objeto Stage. La clase principal de un archivo SWF siempre amplía la clase Sprite (para más información, consulte "Ventajas de la utilización de la lista de visualización" en la página 283).

  Es posible acceder al objeto Stage a través de la propiedad `stage` de ninguna instancia de DisplayObject. Para obtener más información, consulte "Configuración de las propiedades de Stage" en la página 292.

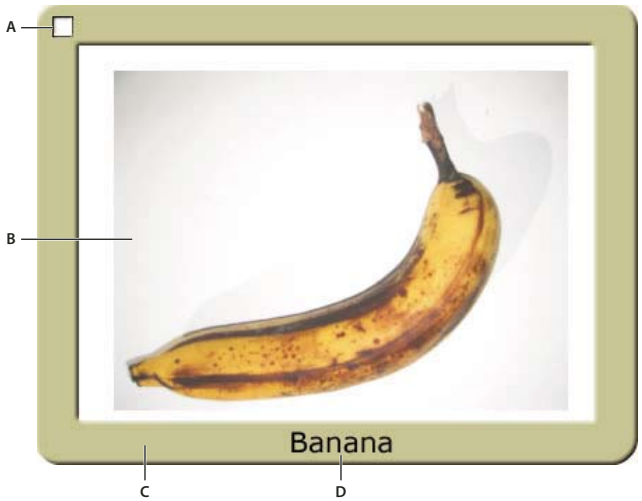
- Objetos de visualización

  En ActionScript 3.0, todos los elementos que aparecen en la pantalla de una aplicación son tipos de *objetos de visualización*. El paquete flash.display incluye una clase DisplayObject. Se trata de una clase base ampliada por diversas otras. Estas distintas clases representan tipos diferentes de objetos de visualización, como formas vectoriales, clips de película o campos de texto, entre otros. Para obtener información general de estas clases, consulte "Ventajas de la utilización de la lista de visualización" en la página 283.

- Contenedores de objetos de visualización

  Los contenedores de objetos de visualización son tipos especiales de objetos de visualización que, además de tener su propia representación visual, pueden contener objetos secundarios que también sean objetos de visualización.

  La clase DisplayObjectContainer es una subclase de la clase DisplayObject. Un objeto DisplayObjectContainer puede contener varios objetos de visualización en su *lista deelementos secundarios*. Por ejemplo, la siguiente ilustración muestra un tipo de objeto DisplayObjectContainer, denominado Sprite, que contiene diversos objetos de visualización:



A. *Un objeto SimpleButton. Este tipo de objeto de visualización tiene distintos estados "Arriba", "Abajo" y "Sobre".* ***B.*** *Un objeto Bitmap. En este caso, el objeto Bitmap se cargó de un archivo JPEG externo a través de un objeto Loader.* ***C.*** *Un objeto Shape. El "marco del cuadro" contiene un rectángulo redondeado que se dibuja en ActionScript. Este objeto Shape tiene aplicado un filtro de sombra.* ***D.*** *Un objeto TextField.*

Cuando se habla de los objetos de visualización, también se hace referencia a los objetos DisplayObjectContainer como *contenedores de objetos de visualización* o simplemente como *contenedores*. Como se ha comentado previamente, el objeto Stage es un contenedor de objeto de visualización.

Aunque todos los objetos de visualización visibles heredan de la clase DisplayObject, el tipo de cada uno de ellos pertenece a una subclase específica de la clase DisplayObject. Por ejemplo, hay una función constructora para la clase Shape o la clase Video, pero no hay ninguna función constructora para la clase DisplayObject.

Tareas comunes de programación de la visualización

Como gran parte de la programación de ActionScript implica crear y manipular elementos visuales, se llevan a cabo numerosas tareas relacionadas con la programación de la visualización. En este capítulo se describen las tareas comunes que se aplican a todos los objetos de visualización:

- Trabajo con la lista de visualización y los contenedores de objetos de visualización
  - Añadir objetos de visualización a la lista de visualización
  - Quitar objetos de la lista de visualización
  - Mover objetos entre contenedores de visualización
  - Mover objetos delante o detrás de otros objetos
- Trabajo con el escenario
  - Definir la velocidad de fotogramas
  - Controlar el ajuste de escala del escenario
  - Trabajo con el modo de pantalla completa
- Gestionar eventos de objetos de visualización
- Colocar objetos de visualización, incluida la creación de interacción de arrastrar y colocar
- Cambiar el tamaño de los objetos de visualización, ajustar su escala y girarlos
- Aplicar modos de mezcla, transformaciones de color y transparencia a los objetos de visualización
- Enmascarar objetos de visualización
- Animar objetos de visualización
- Cargar contenido de visualización externo (como archivos SWF o imágenes)

En capítulos posteriores de este manual se describen otras tareas adicionales para trabajar con objetos de visualización. Se trata de tareas que se aplican a cualquier objeto de visualización o de tareas asociadas a determinados tipos de objetos de visualización:

- Dibujo de gráficos vectoriales con ActionScript en objetos de visualización, descrito en "Utilización de la API de dibujo" en la página 329
- Aplicación de transformaciones geométricas en objetos de visualización, descrito en "Trabajo con la geometría" en la página 351
- Aplicación de efectos gráficos de filtro (como desenfoque, iluminado o sombra, entre otros) en objetos de visualización, descrito en "Aplicación de filtros a objetos de visualización" en la página 363
- Trabajo con características específicas de MovieClip, descrito en "Trabajo con clips de película" en la página 417
- Trabajo con objetos TextField, descrito en "Trabajo con texto" en la página 444
- Trabajo con gráficos de mapas de bits, descrito en "Trabajo con mapas de bits" en la página 495

- Trabajo con elementos de vídeo, descrito en "Trabajo con vídeo" en la página 538

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Alfa: valor del color que representa el grado de transparencia (en realidad, el grado de opacidad) de un color. Por ejemplo, un color con un valor de canal alfa de 60% sólo muestra el 60% de su intensidad total y es un 40% transparente.
- Gráfico de mapa de bits: gráfico que se define en el equipo como una cuadrícula (filas y columnas) de píxeles de colores. Los gráficos de mapa de bits suelen incluir fotos e imágenes similares.
- Modo de fusión: especificación de cómo debe interactuar el contenido de dos imágenes solapadas. Normalmente, una imagen opaca sobre otra imagen simplemente bloquea la imagen de debajo de forma que no esté visible; no obstante, los distintos modos de mezcla hacen que los colores de las imágenes se mezclen de maneras diferentes, por lo que el contenido resultante es una combinación de las dos imágenes.
- Lista de visualización: jerarquía de objetos de visualización que Flash Player y AIR representarán como contenido visible en pantalla. El escenario es la raíz de la lista de visualización y todos los objetos de visualización asociados al escenario o a uno de sus elementos secundarios constituyen la lista de visualización, aunque el objeto no se represente realmente (por ejemplo, si está fuera de los límites del escenario).
- Objeto de visualización: objeto que representa algún tipo de contenido visual en Flash Player o AIR. Sólo se pueden incluir objetos de visualización en la lista de visualización y todas las clases de objetos de visualización son subclases de la clase DisplayObject.
- Contenedor de objetos de visualización: tipo especial de objeto de visualización que puede contener objetos de visualización secundarios además de tener (generalmente) su propia representación visual.
- Clase principal del archivo SWF: clase que define el comportamiento del objeto de visualización más exterior en un archivo SWF, que conceptualmente es la clase para el mismo archivo SWF. Por ejemplo, un archivo SWF creado mediante edición en Flash tiene una "línea de tiempo principal" que contiene todas las demás líneas de tiempo; la clase principal del archivo SWF es la clase de la que la línea de tiempo principal es una instancia.
- Enmascaramiento: técnica para ocultar determinadas partes de una imagen (o a la inversa, para permitir únicamente la visualización de partes determinadas de una imagen). Las partes ocultas de la imagen pasan a ser transparentes, por lo que se puede ver el contenido de debajo. El término en inglés ("masking") está relacionado con la cinta de pintor ("masking tape") que se aplica para evitar pintar donde no hay que pintar.
- Escenario: contenedor visual que constituye la base o el fondo de todo el contenido visual de un archivo SWF.
- Transformación: ajuste de una característica visual de un gráfico, como girar el objeto, modificar su escala, sesgar o distorsionar su forma, o bien, modificar su color.
- Gráfico vectorial: gráfico definido en el equipo como líneas y formas dibujadas con características específicas (como grosor, longitud, tamaño, ángulo y posición).

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Debido a que este capítulo se centra en crear y manipular contenido visual, prácticamente todos los listados de código crean objetos visuales y los muestran en pantalla; para probar los ejemplos es necesario ver el resultado en Flash Player o AIR, en lugar de ver los valores de las variables como en los capítulos anteriores. Para probar los listados de código de este capítulo:

1. Cree un documento vacío utilizando la herramienta de edición de Flash.

2 Seleccione un fotograma clave en la línea de tiempo.

3 Abra el panel Acciones y copie el listado de código en el panel Script.

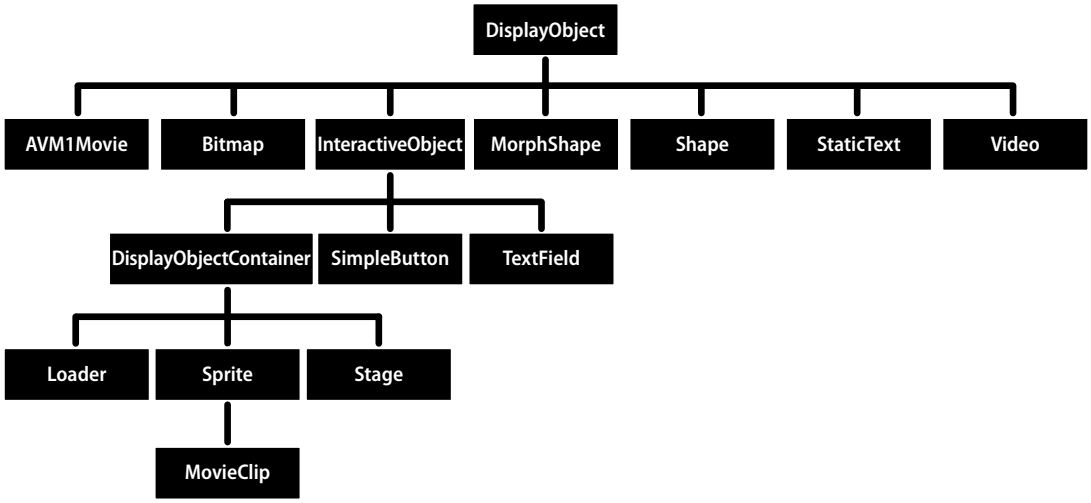
4 Ejecute el programa seleccionando Control > Probar película.

Verá los resultados del código en la pantalla y las llamadas a la función `trace()` se mostrarán en el panel Salida.

Las técnicas para probar el código de ejemplo se explican más detalladamente en "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

Clases principales de visualización

El paquete flash.display de ActionScript 3.0 contiene clases de objetos visuales que pueden aparecer en Flash Player o AIR. En la siguiente ilustración se muestran las relaciones entre las subclases de estas clases principales de objetos de visualización.



En la ilustración se muestra la herencia de clases de las clases principales de visualización. Debe tenerse en cuenta que algunas de estas clases, en concreto, StaticText, TextField y Video, no se encuentran en el paquete flash.display pero también heredan de la clase DisplayObject.

Todas las clases que amplían la clase DisplayObject heredan sus métodos y propiedades. Para obtener más información, consulte "Propiedades y métodos de la clase DisplayObject" en la página 286.

Pueden crearse instancias de objetos de las siguientes clases contenidas en el paquete flash.display:

- Bitmap: la clase Bitmap se usa para definir objetos de mapa de bits cargados de archivos externos o representados a través de ActionScript. Para cargar mapas de bits de archivos externos, se puede usar la clase Loader. Se pueden cargar archivos GIF, JPG o PNG. También es posible crear un objeto BitmapData con datos personalizados y crear a continuación un objeto Bitmap que utilice dichos datos. Se pueden usar los métodos de la clase BitmapData para modificar mapas de bits, tanto si se cargan como si se crean en ActionScript. Para obtener más información, consulte "Carga de objetos de visualización" en la página 320 y "Trabajo con mapas de bits" en la página 495.
- Loader: la clase Loader se usa para cargar activos externos (archivos SWF o gráficos). Para obtener más información, consulte "Carga dinámica de contenido de visualización" en la página 320.

- Shape: la clase Shape se usa para crear gráficos vectoriales como rectángulos, líneas, círculos, etc. Para obtener más información, consulte "Utilización de la API de dibujo" en la página 329.
- SimpleButton. Un objeto SimpleButton es la representación de ActionScript de un símbolo de botón creado en la herramienta de edición de Flash. Una instancia de SimpleButton dispone de cuatro estados de botón: up (arriba), down (abajo), over (sobre) y hit test (prueba de zona activa) (el área que responde a eventos de teclado y ratón).
- Sprite: un objeto Sprite puede contener gráficos propios y también objetos de visualización secundarios. La clase Sprite amplía la clase DisplayObjectContainer. Para obtener más información, consulte "Trabajo con contenedores de objetos de visualización" en la página 287 y "Utilización de la API de dibujo" en la página 329.
- MovieClip: un objeto MovieClip es la representación en ActionScript de un símbolo de clip de película creado en la herramienta de edición de Flash. En la práctica, un MovieClip es similar a un objeto Sprite, excepto en que tiene además una línea de tiempo. Para obtener más información, consulte "Trabajo con clips de película" en la página 417.

Las siguientes clases, que no se encuentran en el paquete flash.display, son subclases de la clase DisplayObject:

- La clase TextField, incluida en el paquete flash.text, es un objeto de visualización para mostrar e introducir texto. Para obtener más información, consulte "Trabajo con texto" en la página 444.
- La clase Video, incluida en el paquete flash.media, es el objeto de visualización que se utiliza para mostrar archivos de vídeo. Para obtener más información, consulte "Trabajo con vídeo" en la página 538.

Las siguientes clases del paquete flash.display amplían la clase DisplayObject, pero no es posible crear instancias de las mismas. En lugar de eso, actúan como clases principales de otros objetos de visualización y combinan la funcionalidad común en una sola clase.

- AVM1Movie: la clase AVM1Movie se usa para representar los archivos SWF cargados que se crearon en ActionScript 1.0 y 2.0.
- DisplayObjectContainer: las clases Loader, Stage, Sprite y MovieClip amplían la clase DisplayObjectContainer. Para obtener más información, consulte "Trabajo con contenedores de objetos de visualización" en la página 287.
- InteractiveObject: InteractiveObject es la clase base de todos los objetos y se utiliza para interactuar con el ratón y el teclado. Los objetos SimpleButton, TextField, Loader, Sprite, Stage y MovieClip son subclases de la clase InteractiveObject. Para obtener más información sobre la creación de interacción de teclado y ratón, consulte "Captura de entradas del usuario" en la página 610.
- MorphShape: estos objetos se crean al crear una interpolación de forma en la herramienta de edición de Flash. No es posible crear instancias de estos objetos con ActionScript pero se puede acceder a ellos desde la lista de visualización.
- Stage: la clase Stage amplía la clase DisplayObjectContainer. Hay una instancia de Stage por aplicación y se sitúa en lo más alto de la jerarquía de la lista de visualización. Para acceder a Stage, debe usarse la propiedad `stage` de cualquier instancia de DisplayObject. Para obtener más información, consulte "Configuración de las propiedades de Stage" en la página 292.

Además, la clase StaticText del paquete flash.text amplía la clase DisplayObject, pero no es posible crear una instancia de ella en el código. Los campos de texto estático se crean únicamente en Flash.

Ventajas de la utilización de la lista de visualización

En ActionScript 3.0, hay clases independientes para los distintos tipos de objetos de visualización. En ActionScript 1.0 y 2.0, muchos de los objetos del mismo tipo se incluyen en una clase: MovieClip.

Esta individualización de clases y la estructura jerárquica de las listas de visualización presentan las siguientes ventajas:

- Mayor eficacia de representación y disminución del uso de memoria
- Mejor administración de profundidad
- Recorrido completo de la lista de visualización
- Objetos de visualización fuera de la lista
- Creación más sencilla de subclases de objetos de visualización

Mayor eficacia de representación y tamaños de archivo más pequeños

En ActionScript 1.0 y 2.0, sólo era posible dibujar formas en un objeto MovieClip. En ActionScript 3.0, hay clases de objetos de visualización más sencillas en las que se pueden dibujar objetos. Dado que estas clases de objetos de visualización de ActionScript 3.0 no incluyen el conjunto completo de métodos y propiedades que contiene un objeto MovieClip, requieren menos recursos de memoria y de procesador.

Por ejemplo, cada objeto MovieClip incluye propiedades de la línea de tiempo del clip de película, pero un objeto Shape no. Las propiedades para administrar la línea de tiempo pueden consumir muchos recursos de memoria y de procesador. En ActionScript 3.0, el uso del objeto Shape permite mejorar el rendimiento. El objeto Shape tiene una sobrecarga menor que el objeto MovieClip, que es más complejo. Flash Player y AIR no necesitan administrar las propiedades de MovieClip que no se usan, lo cual mejora la velocidad y disminuye la memoria que utiliza el objeto.

Mejor administración de profundidad

En ActionScript 1.0 y 2.0, la profundidad se administraba a través de un esquema lineal de administración de profundidad y de métodos como `getNextHighestDepth()`.

ActionScript 3.0 incluye la clase DisplayObjectContainer, que tiene más métodos y propiedades útiles para administrar la profundidad de los objetos de visualización.

En ActionScript 3.0, al mover un objeto de visualización a una nueva posición en la lista de elementos secundarios de una instancia de DisplayObjectContainer, los demás elementos secundarios del contenedor de objeto de visualización se vuelven a colocar automáticamente y reciben la asignación de las posiciones adecuadas en el índice de elementos secundarios del contenedor de objeto de visualización.

Además, en ActionScript 3.0 siempre es posible descubrir todos los objetos secundarios de cualquier contenedor de objeto de visualización. Cada instancia de DisplayObjectContainer tiene una propiedad `numChildren`, que muestra el número de elementos secundarios en el contenedor de objeto de visualización. Y como la lista de elementos secundarios de un contenedor de objeto de visualización es siempre una lista con índice, se puede examinar cada objeto de la lista desde la posición 0 hasta la última posición del índice (`numChildren - 1`). Esto no era posible con los métodos y propiedades de un objeto MovieClip en ActionScript 1.0 y 2.0.

En ActionScript 3.0, se puede recorrer fácilmente la lista de visualización de forma secuencial; no hay huecos en los números de índice de una lista de elementos secundarios de un contenedor de objeto de visualización. Recorrer la lista de visualización y administrar la profundidad de los objetos es mucho más sencillo que lo era en ActionScript 1.0 y 2.0. En ActionScript 1.0 y 2.0, un clip de película podía incluir objetos con huecos intermitentes en el orden de profundidad, lo que podía dificultar el recorrido de la lista de objetos. En ActionScript 3.0, cada lista de elementos secundarios de un contenedor de objeto de visualización se almacena en caché internamente como un conjunto, lo que permite realizar búsquedas rápidamente (por índice). Reproducir indefinidamente todos los elementos secundarios de un contenedor de objeto de visualización es también muy rápido.

En ActionScript 3.0, también se puede acceder a los elementos secundarios de un contenedor de objeto de visualización a través del método `getChildByName()` de la clase DisplayObjectContainer.

Recorrido completo de la lista de visualización

En ActionScript 1.0 y 2.0, no era posible acceder a algunos objetos que se dibujaban en la herramienta de edición de Flash como, por ejemplo, las formas vectoriales. En ActionScript 3.0, es posible acceder a todos los objetos de la lista de visualización creados con ActionScript o con la herramienta de edición de Flash. Para obtener información, consulte "Recorrido de la lista de visualización" en la página 291.

Objetos de visualización fuera de la lista

En ActionScript 3.0, se pueden crear objetos de visualización que no estén incluidos en la lista de visualización visible. Se conocen como objetos de visualización *fuera de la lista*. Un objeto de visualización sólo se añade a la lista de visualización visible cuando se llama al método `addChild()` o `addChildAt()` de una instancia de DisplayObjectContainer que ya se haya añadido a la lista de visualización.

Se pueden utilizar los objetos de visualización fuera de la lista para ensamblar objetos de visualización complejos, como los que tienen contenedores de objetos de visualización con varios objetos de visualización. Mantener los objetos de visualización fuera de la lista permite ensamblar objetos complicados sin tener que usar el tiempo de procesamiento para representar estos objetos de visualización. Cuando se necesita un objeto de visualización fuera de la lista, se puede añadir a la lista de visualización. Además, se puede mover un elemento secundario de un contenedor de objeto de visualización dentro y fuera de la lista de visualización, y a cualquier posición que se elija en la lista de visualización.

Creación más sencilla de subclases de objetos de visualización

En ActionScript 1.0 y 2.0, a menudo se añadían objetos MovieClip nuevos a un archivo SWF para crear formas básicas o para mostrar mapas de bits. En ActionScript 3.0, la clase DisplayObject incluye numerosas subclases incorporadas, como Shape y Bitmap. Como las clases de ActionScript 3.0 están más especializadas para determinados tipos de objetos, resulta más sencillo crear subclases básicas de las clases incorporadas.

Por ejemplo, para dibujar un círculo en ActionScript 2.0, se podía crear una clase CustomCircle que ampliara la clase MovieClip al crear una instancia de un objeto de la clase personalizada. Sin embargo, esa clase incluiría además varias propiedades y métodos de la clase MovieClip (por ejemplo, `totalFrames`) que no se aplican a la clase. Sin embargo, en ActionScript 3.0 es posible crear una clase CustomCircle que amplíe el objeto Shape y que, como tal, no incluya las propiedades y métodos no relacionados contenidos en la clase MovieClip. El código siguiente muestra un ejemplo de una clase CustomCircle:

```
import flash.display.*;

public class CustomCircle extends Shape
{
    var xPos:Number;
    var yPos:Number;
    var radius:Number;
    var color:uint;
    public function CustomCircle(xInput:Number,
                                 yInput:Number,
                                 rInput:Number,
                                 colorInput:uint)
    {
        xPos = xInput;
        yPos = yInput;
        radius = rInput;
        color = colorInput;
        this.graphics.beginFill(color);
        this.graphics.drawCircle(xPos, yPos, radius);
    }
}
```

Trabajo con objetos de visualización

Una vez comprendidos los conceptos básicos del escenario, los objetos de visualización, los contenedores de objetos de visualización y la lista de visualización, esta sección proporciona información más específica relativa a la utilización de los objetos de visualización en ActionScript 3.0.

Propiedades y métodos de la clase DisplayObject

Todos los objetos de visualización son subclases de la clase DisplayObject y, como tales, heredan las propiedades y métodos de la clase DisplayObject. Las propiedades heredadas son propiedades básicas que se aplican a todos los objetos de visualización. Por ejemplo, cada objeto de visualización tiene una propiedad `x` y una propiedad `y` que especifican la posición del objeto en su contenedor de objeto de visualización.

No se puede crear una instancia de DisplayObject con el constructor de la clase DisplayObject. Se debe crear otro tipo de objeto (un objeto que sea una subclase de la clase DisplayObject), como Sprite, para crear una instancia de un objeto con el operador `new`. Además, si se desea crear una clase personalizada de un objeto de visualización, se debe crear una subclase de una de las subclases del objeto de visualización que tenga una función constructora que pueda utilizarse (por ejemplo, la clase Shape o la clase Sprite). Para obtener más información, consulte la descripción de la clase DisplayObject en la Referencia del lenguaje y componentes ActionScript 3.0.

Añadir objetos de visualización a la lista de visualización

Cuando se crea una instancia de un objeto de visualización, no aparecerá en pantalla (en el escenario) hasta que se añada la instancia del objeto de visualización a un contenedor de objeto de visualización de la lista de visualización. Por ejemplo, en el código siguiente, el objeto TextField `myText` no sería visible si se omitiera la última línea de código. En la última línea de código, la palabra clave `this` debe hacer referencia a un contenedor de objeto de visualización que ya se haya añadido a la lista de visualización.

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

Cuando se añade un elemento visual al escenario, dicho elemento se convierte en un *elemento secundario* del objeto Stage. El primer archivo SWF cargado en una aplicación (por ejemplo, el que se incorpora en una página HTML) se añade automáticamente como elemento secundario del escenario. Puede ser un objeto de cualquier tipo que amplíe la clase Sprite.

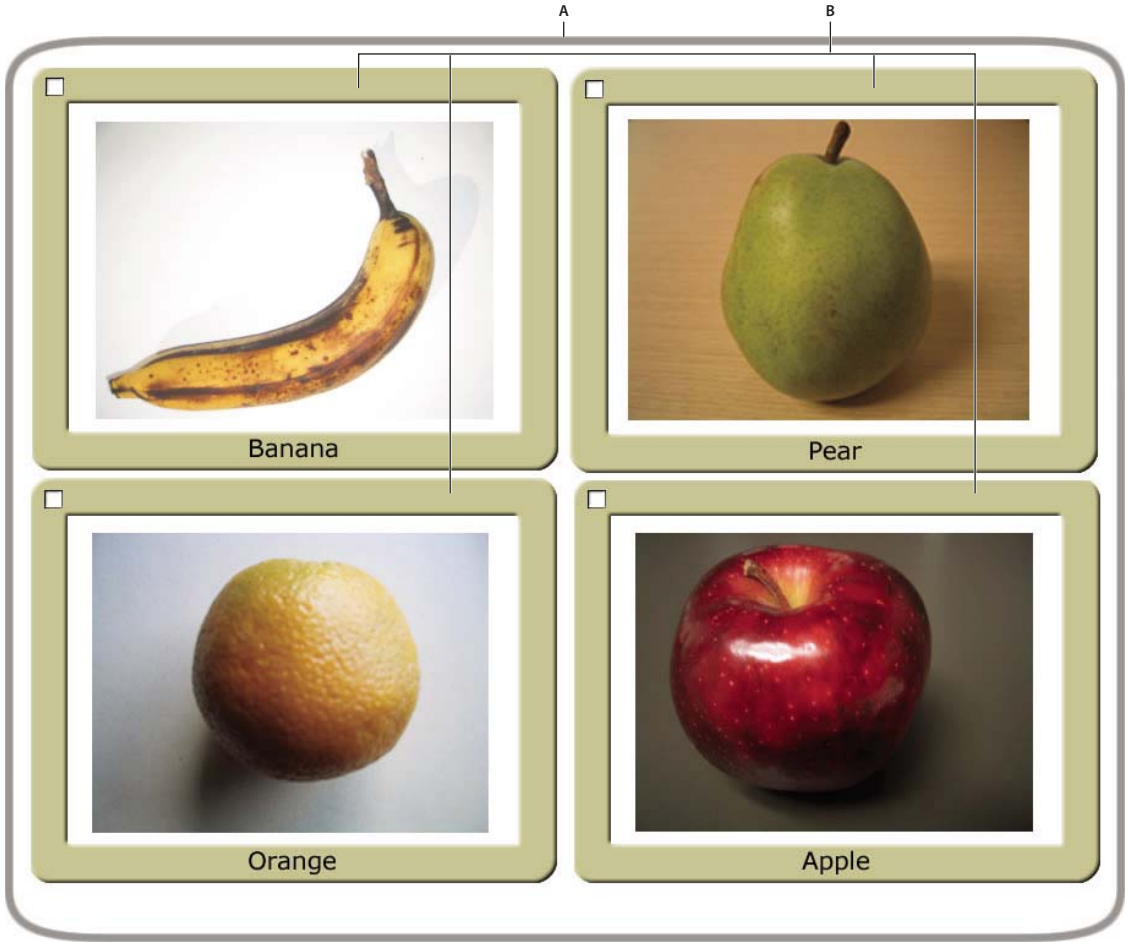
Los objetos de visualización que se crean *sin* usar ActionScript (por ejemplo, añadiendo una etiqueta MXML en Adobe Flex Builder 3 o colocando un elemento en el escenario de Flash) se añaden a la lista de visualización. Aunque estos objetos de visualización no se añadan mediante ActionScript, sí se puede acceder a ellos a través de ActionScript. Por ejemplo, el código siguiente ajusta la anchura de un objeto denominado `button1`, que se añadió en la herramienta de edición (no a través de ActionScript):

```
button1.width = 200;
```

Trabajo con contenedores de objetos de visualización

Si se elimina un objeto DisplayObjectContainer de la lista de visualización o si se mueve o transforma de algún otro modo, también se elimina, mueve o transforma cada objeto de visualización de DisplayObjectContainer.

Un contenedor de objeto de visualización es por sí mismo un tipo de objeto de visualización; puede añadirse a otro contenedor de objeto de visualización. Por ejemplo, en la imagen siguiente se muestra un contenedor de objeto de visualización, `pictureScreen`, que contiene una forma de contorno y otros cuatro contenedores de objetos de visualización (del tipo PictureFrame):



***A.** Una forma que define el borde del contenedor de objeto de visualización pictureScreen **B.** Cuatro contenedores de objetos de visualización que son elementos secundarios del objeto pictureScreen*

Para que un objeto de visualización aparezca en la lista de visualización, debe añadirse a un contenedor de objeto de visualización incluido en la lista de visualización. Para ello, se utiliza el método `addChild()` o el método `addChildAt()` del objeto contenedor. Por ejemplo, sin la línea final del código siguiente, no se mostraría el objeto `myTextField`:

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

En este ejemplo de código, `this.root` señala al contenedor de objeto de visualización MovieClip que contiene el código. En el código real, se puede especificar un contenedor distinto.

Se debe usar el método `addChildAt()` para añadir el elemento secundario a una posición específica de la lista secundaria del contenedor de objeto de visualización. Estas posiciones del índice basado en cero de la lista de elementos secundarios se refieren a la organización en capas (orden de delante a atrás) de los objetos de visualización. Por ejemplo, observe los tres objetos de visualización siguientes. Cada objeto se creó a partir de una clase personalizada denominada Ball.

La organización en capas de estos objetos de visualización en el contenedor puede ajustarse con el método `addChildAt()`. Por ejemplo, considérese el fragmento de código siguiente:

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

Después de ejecutar este código, los objetos de visualización se colocan del siguiente modo en el objeto DisplayObjectContainer `container`. Observe la organización en capas de los objetos.

Para volver a colocar un objeto en la parte superior de la lista de visualización, simplemente hay que volver a añadirlo a la lista. Por ejemplo, después del código anterior, para mover `ball_A` a la parte superior de la pila, debe utilizarse esta línea de código:

```
container.addChild(ball_A);
```

Este código quita `ball_A` de su ubicación en la lista de visualización de `container` y vuelve a añadirlo a la parte superior de la lista, por lo que finalmente se mueve a la parte superior de la pila.

Puede usarse el método `getChildAt()` para verificar el orden de las capas de los objetos de visualización. El método `getChildAt()` devuelve objetos secundarios de un contenedor basándose en el número de índice que se pasa. Por ejemplo, el código siguiente muestra nombres de objetos de visualización en distintas posiciones de la lista de elementos secundarios del objeto DisplayObjectContainer `container`:

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

Si se quita un objeto de visualización de la lista de elementos secundarios del contenedor principal, los elementos de niveles superiores de la lista descienden una posición en el índice de elementos secundarios. Por ejemplo, siguiendo con el código anterior, el código siguiente muestra cómo el objeto de visualización que estaba en la posición 2 del objeto DisplayObjectContainer `container` se mueve a la posición 1 si se quita un objeto de visualización situado en un nivel inferior de la lista de elementos secundarios:

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

Los métodos `removeChild()` y `removeChildAt()` no eliminan completamente una instancia de objeto de visualización. Simplemente la quitan de la lista de elementos secundarios del contenedor. Otra variable podrá seguir haciendo referencia a la instancia. (Para eliminar completamente un objeto, debe usarse el operador `delete`.)

Dado que un objeto de visualización sólo tiene un contenedor principal, se puede añadir una instancia de un objeto de visualización a un solo contenedor de objeto de visualización. Por ejemplo, el código siguiente muestra que el objeto de visualización `tf1` sólo puede existir en un contenedor (en este caso, un objeto Sprite, que amplía la clase DisplayObjectContainer):

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // text 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // text 1
```

Si se añade un objeto de visualización de un contenedor de objeto de visualización a otro contenedor de objeto de visualización, se elimina de la lista de elementos secundarios del primer contenedor de objeto de visualización.

Además de los métodos anteriormente descritos, la clase DisplayObjectContainer define varios métodos para trabajar con objetos de visualización secundarios, entre los que se encuentran:

- `contains()`: determina si un objeto de visualización es un elemento secundario de DisplayObjectContainer.
- `getChildByName()`: recupera un objeto de visualización por el nombre.
- `getChildIndex()`: devuelve la posición de índice de un objeto de visualización.
- `setChildIndex()`: cambia la posición de un objeto de visualización secundario.
- `swapChildren()`: intercambia el orden de delante a atrás de dos objetos de visualización.

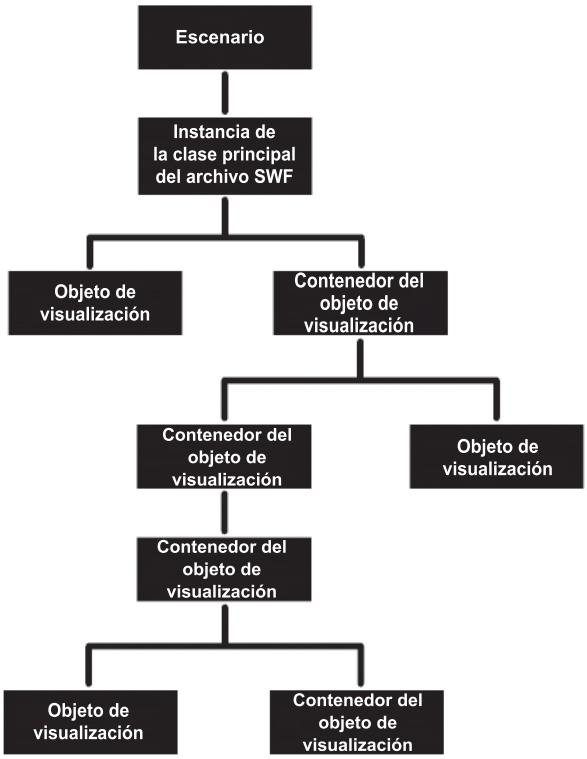
- `swapChildrenAt()`: intercambia el orden de delante a atrás de dos objetos de visualización, especificados por sus valores de índice.

Para obtener más información, consulte las entradas correspondientes de la Referencia del lenguaje y componentes ActionScript 3.0.

Cabe recordar que un objeto de visualización que esté fuera de la lista de visualización (es decir, que no esté incluido en un contenedor de objeto de visualización secundario del objeto Stage) se denomina objeto de visualización *fuera de la lista*.

Recorrido de la lista de visualización

Tal como se ha visto, la lista de visualización tiene una estructura de árbol. En la parte superior del árbol está el objeto Stage, que puede contener varios objetos de visualización. Estos objetos de visualización, que son al mismo tiempo contenedores de objetos de visualización, pueden contener otros objetos de visualización o contenedores de objetos de visualización.



La clase DisplayObjectContainer incluye propiedades y métodos para recorrer la lista de visualización, mediante las listas de elementos secundarios de los contenedores de objetos de visualización. Por ejemplo, el código siguiente añade dos objetos de visualización, `title` y `pict`, al objeto `container` (que es un objeto Sprite y la clase Sprite amplía la clase DisplayObjectContainer):

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

El método `getChildAt()` devuelve el elemento secundario de la lista de visualización en una posición de índice específica:

```
trace(container.getChildAt(0) is TextField); // true
```

También se puede acceder a los objetos secundarios por el nombre. Todos los objetos de visualización tienen una propiedad de nombre y, si no se asigna, Flash Player o AIR asignan un valor predeterminado como, por ejemplo, `"instance1"`. Por ejemplo, el código siguiente muestra cómo utilizar el método `getChildByName()` para acceder a un objeto de visualización secundario con el nombre `"banana loader"`:

```
trace(container.getChildByName("banana loader") is Loader); // true
```

Si se utiliza el método `getChildByName()` el resultado es más lento que si se usa el método `getChildAt()`.

Como un contenedor de objeto de visualización puede contener otros contenedores de objetos de visualización como objetos secundarios en su lista de visualización, se puede recorrer toda la lista de visualización de la aplicación como un árbol. Por ejemplo, en el fragmento de código anterior, cuando finaliza la operación de carga del objeto Loader `pict`, el objeto `pict` tendrá cargado un objeto de visualización secundario, que es el mapa de bits. Para acceder a este objeto de visualización de mapa de bits, se puede escribir `pict.getChildAt(0)`. También se puede escribir `container.getChildAt(0).getChildAt(0)` (porque `container.getChildAt(0) == pict`).

La función siguiente proporciona una salida `trace()` con sangría de la lista de visualización desde un contenedor de objeto de visualización:

```
function traceDisplayList(container:DisplayObjectContainer,indentString:String = ""):void
{
    var child:DisplayObject;
    for (var i:uint=0; i < container.numChildren; i++)
    {
        child = container.getChildAt(i);
        trace(indentString, child, child.name);
        if (container.getChildAt(i) is DisplayObjectContainer)
        {
            traceDisplayList(DisplayObjectContainer(child), indentString + "")
        }
    }
}
```

Configuración de las propiedades de Stage

La clase Stage sustituye la mayoría de las propiedades y métodos de la clase DisplayObject. Si se llama a una de estas propiedades o métodos sustituidos, Flash Player o AIR emiten una excepción. Por ejemplo, el objeto Stage no tiene propiedades `x` ni `y` porque su posición es fija como el contenedor principal de la aplicación. Las propiedades `x` e `y` hacen referencia a la posición de un objeto de visualización con respecto a su contenedor; como Stage no se incluye en ningún otro contenedor de objeto de visualización, estas propiedades no se aplican.

Nota: *algunas propiedades y métodos de la clase Stage sólo están disponibles para los objetos de visualización que se encuentran en el mismo entorno limitado de seguridad que el primer archivo SWF cargado. Para obtener información, consulte "Seguridad del objeto Stage" en la página 733.*

Control de la velocidad de fotogramas de reproducción

La propiedad `framerate` de la clase Stage se utiliza para definir la velocidad de fotogramas de todos los archivos SWF cargados en la aplicación. Para obtener más información, consulte la Referencia del lenguaje y componentes ActionScript 3.0.

Controlar el ajuste de escala del escenario

Cuando la parte de la pantalla que representan Flash Player o AIR cambia de tamaño, las aplicaciones ajustan automáticamente el contenido del escenario para realizar una compensación. La propiedad `scaleMode` de la clase Stage determina cómo se ajustará el contenido del escenario. Esta propiedad se puede establecer en cuatro valores distintos, definidos como constantes en la clase flash.display.StageScaleMode.

Para tres de los valores de `scaleMode` (`StageScaleMode.EXACT_FIT`, `StageScaleMode.SHOW_ALL` y `StageScaleMode.NO_BORDER`), Flash Player y AIR escalarán el contenido de Stage para que se ajuste dentro de sus límites. Las tres opciones difieren a la hora de determinar el modo en que se lleva a cabo la escala:

- `StageScaleMode.EXACT_FIT` ajusta proporcionalmente la escala del archivo SWF.
- `StageScaleMode.SHOW_ALL` determina si aparece un contorno, como las barras de color negro que aparecen al ver una película de pantalla panorámica en un televisor normal.
- `StageScaleMode.NO_BORDER` determina si se puede recortar parcialmente el contenido o no.

Como alternativa, si `scaleMode` se establece en `StageScaleMode.NO_SCALE`, el contenido de Stage mantendrá su tamaño definido cuando el usuario cambie el tamaño de la ventana de Flash Player o AIR. Éste es el único modo de escala que permite utilizar las propiedades `Width` y `Height` de la clase Stage para determinar las dimensiones en píxeles reales de la ventana de cuyo tamaño se ha ajustado. (En los otros modos de escala, las propiedades `stageWidth` y `stageHeight` siempre reflejan la anchura y la altura originales del archivo SWF.) Además, si se establece `scaleMode` en `StageScaleMode.NO_SCALE` y se ajusta el tamaño del archivo SWF, se distribuye el evento `resize` de la clase Stage, lo que permite realizar los ajustes necesarios.

Por consiguiente, si establece `scaleMode` en `StageScaleMode.NO_SCALE`, tendrá mayor control sobre el ajuste del contenido de la pantalla al cambiar el tamaño de la ventana. Por ejemplo, en un archivo SWF que contiene un vídeo y una barra de control, puede que desee que la barra de control mantenga el mismo tamaño cuando se cambie el tamaño del escenario y modificar únicamente el tamaño de la ventana de vídeo para adaptarla al cambio de tamaño del escenario. Esto se ilustra en el siguiente ejemplo:

```
// videoScreen is a display object (e.g. a Video instance) containing a
// video; it is positioned at the top-left corner of the Stage, and
// it should resize when the SWF resizes.

// controlBar is a display object (e.g. a Sprite) containing several
// buttons; it should stay positioned at the bottom-left corner of the
// Stage (below videoScreen) and it should not resize when the SWF
// resizes.

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = videoScreen.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;

function resizeDisplay(event:Event):void
{
    var swfWidth:int = swfStage.stageWidth;
    var swfHeight:int = swfStage.stageHeight;

    // Resize the video window.
    var newVideoHeight:Number = swfHeight - controlBar.height;
    videoScreen.height = newVideoHeight;
    videoScreen.scaleX = videoScreen.scaleY;

    // Reposition the control bar.
    controlBar.y = newVideoHeight;
}

swfStage.addEventListener(Event.RESIZE, resizeDisplay);
```

Trabajo con el modo de pantalla completa

El modo de pantalla completa permite establecer un escenario de película para rellenar el monitor completo de un espectador sin ningún borde de contenedor o menú. La propiedad `displayState` de la clase Stage se utiliza para activar y desactivar el modo de pantalla completa para un archivo SWF. La propiedad `displayState` puede establecerse en uno de los valores definidos por las constantes de la clase flash.display.StageDisplayState. Para activar el modo de pantalla completa, debe establecerse la propiedad `displayState` en `StageDisplayState.FULL_SCREEN`:

```
stage.displayState = StageDisplayState.FULL_SCREEN;
```

En Flash Player, el modo de pantalla completa sólo puede iniciarse a través de ActionScript como respuesta a un clic del ratón (incluido el clic con el botón derecho) o una pulsación de tecla. El contenido de AIR que se ejecuta en el entorno limitado de seguridad de la aplicación no requiere que se indique el modo de pantalla completa como respuesta a un gesto del usuario.

Para salir del modo de pantalla completa, debe establecerse la propiedad `displayState` en `StageDisplayState.NORMAL`.

```
stage.displayState = StageDisplayState.NORMAL;
```

Asimismo, el usuario puede optar por salir del modo de pantalla completa cambiando a una ventana diferente o utilizando una de las distintas combinaciones de teclas: tecla Esc (todas las plataformas), Control-W (Windows), Comando-W (Mac) o Alt-F4 (Windows).

Activación del modo de pantalla completa en Flash Player

Para activar el modo de pantalla completa en un archivo SWF incorporado en una página HTML, el código HTML para incorporar Flash Player debe incluir una etiqueta `param` y un atributo `embed` con el nombre `allowFullScreen` y el valor `true`, como en el siguiente código:

```
<object>
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowfullscreen="true" />
</object>
```

En la herramienta de edición de Flash, seleccione Archivo -> Configuración de publicación y en el cuadro de diálogo Configuración de publicación, en la ficha HTML, seleccione la plantilla Sólo Flash - Permitir pantalla completa.

En Flex, asegúrese de que la plantilla HTML incluya las etiquetas `<object>` y `<embed>` que son compatibles con la pantalla completa.

Si se utiliza JavaScript en una página Web para generar las etiquetas de incorporación de SWF, se debe modificar JavaScript para añadir el atributo/etiqueta de parámetro `allowFullScreen`. Por ejemplo, si la página HTML utiliza la función `AC_FL_RunContent()` (que se usa en páginas HTML generadas tanto en Flex Builder como en Flash), se debe añadir el parámetro `allowFullScreen` a dicha llamada de función, del siguiente modo:

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

Esto no se aplica a los archivos SWF que se ejecutan en el reproductor Flash Player autónomo.

Nota: *si el modo de ventana (wmode en HTML) se establece en Opaco sin ventanas (opaco) o Transparente sin ventanas (transparente), la ventana de pantalla completa siempre es opaca.*

También existen limitaciones relacionadas con la seguridad al utilizar el modo de pantalla completa de Flash Player en un navegador. Estas limitaciones se describen en el capítulo "Seguridad de Flash Player" en la página 714.

Escala y tamaño del escenario en pantalla completa

Las propiedades `Stage.fullScreenHeight` y `Stage.fullScreenWidth` devuelven la altura y la anchura del monitor que se utiliza cuando se pasa al tamaño de pantalla completa, si ese estado se introduce de forma inmediata. Estos valores pueden ser incorrectos si el usuario tiene la oportunidad de mover el navegador de un monitor a otro después de recuperar estos valores pero antes de pasar al modo de pantalla completa. Si se recuperan estos valores en el mismo controlador de eventos en el que se estableció la propiedad `Stage.displayState` como `StageDisplayState.FULL_SCREEN`, los valores son correctos. Para usuarios de varios monitores, el contenido de SWF sólo se expandirá para llenar un monitor. Flash Player y AIR usan una métrica para determinar el monitor que contiene la mayor parte del archivo SWF y utiliza dicho monitor en el modo de pantalla completa. Las propiedades fullScreenHeight y fullScreenWidth sólo reflejan el tamaño del monitor utilizado para el modo de pantalla completa. Para obtener más información, consulte `Stage.fullScreenHeight` y `Stage.fullScreenWidth` en el manual Referencia del lenguaje y componentes ActionScript 3.0.

El comportamiento de ajuste de escala del escenario en el modo de pantalla completa es el mismo que en el modo normal; el ajuste de escala se controla con la propiedad `scaleMode` de la clase Stage. Si la propiedad `scaleMode` se establece en `StageScaleMode.NO_SCALE`, las propiedades `stageWidth` y `stageHeight` de Stage cambian para reflejar el tamaño del área de la pantalla ocupada por el archivo SWF (toda la pantalla en este caso); si se visualiza en el navegador, el parámetro HTML controla la configuración.

Se puede utilizar el evento `fullScreen` de la clase Stage para detectar cuándo se activa y desactiva el modo de pantalla completa, y para responder ante ello. Por ejemplo, puede ser que se desee volver a colocar, añadir o quitar elementos de la pantalla al activar o desactivar el modo de pantalla completa, como en el ejemplo:

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
    if (event.fullScreen)
    {
        // Remove input text fields.
        // Add a button that closes full-screen mode.
    }
    else
    {
        // Re-add input text fields.
        // Remove the button that closes full-screen mode.
    }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

Tal y como se muestra en este código, el objeto del evento `fullScreen` es una instancia de la clase flash.events.FullScreenEvent, que incluye una propiedad `fullScreen` que indica si el modo de pantalla completa está activado (`true`) o no (`false`).

Compatibilidad con el teclado en modo de pantalla completa

Cuando Flash Player se ejecuta en un navegador, todo el código ActionScript relacionado con el teclado, como los eventos de teclado y la introducción de texto en instancias de TextField, se desactiva en modo de pantalla completa. Las excepciones (teclas que permanecen activadas) son:

- Determinadas teclas que no afectan a la impresión, como las teclas de flecha, la barra espaciadora o el tabulador
- Métodos abreviados de teclado que cierran el modo de pantalla completa: Esc (Windows y Mac), Control-W (Windows), Comando-W (Mac) y Alt-F4

Estas restricciones no se aplican a contenido de SWF ejecutado en el reproductor autónomo de Flash Player o en AIR. AIR admite un modo interactivo de pantalla completa que permite la entrada del teclado.

Escala de hardware en modo de pantalla completa

La propiedad `fullScreenSourceRect` de la clase Stage se puede utilizar para establecer que Flash Player o AIR escalen una región específica del escenario al modo de pantalla completa. La escala de hardware en Flash Player y AIR, en caso de estar disponible, se activa en la tarjeta gráfica y de vídeo del equipo y suele mostrar el contenido con mayor rapidez que la escala de software.

Para aprovechar la escala de hardware, debe establecer todo el escenario o parte del mismo en modo de pantalla completa. El siguiente código ActionScript 3.0 establece todo el escenario en modo de pantalla completa:

```
import flash.geom.*;
{
    stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Cuando esta propiedad se establece en un rectángulo válido y la propiedad `displayState` se define en modo de pantalla completa, Flash Player y AIR aplican escala al área especificada. El tamaño real del escenario expresado en píxeles dentro de ActionScript no cambia. Flash Player y AIR aplican un límite mínimo para el tamaño del rectángulo de modo que se ajuste al mensaje estándar "Press Esc to exit full-screen mode" (Presione Esc para salir del modo de pantalla completa). Este límite suele ser de 260 x 30 píxeles, pero puede variar según la plataforma y la versión de Flash Player.

La propiedad `fullScreenSourceRect` sólo se puede definir cuando Flash Player o AIR no estén en modo de pantalla completa. Para utilizar esta propiedad correctamente, primero debe establecerse y, seguidamente, definir la propiedad `displayState` en modo de pantalla completa.

Para activar la escala, establezca la propiedad `fullScreenSourceRect` en un objeto rectangle.

```
stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
```

Para desactivar la escala, establezca la propiedad `fullScreenSourceRect` en `null`.

```
stage.fullScreenSourceRect = null;
```

Para aprovechar íntegramente las funciones de aceleración de hardware con Flash Player, actívela mediante el cuadro de diálogo Configuración de Flash Player. Para cargar el cuadro de diálogo, haga clic con el botón derecho del ratón (Windows) o presione Control y haga clic (Mac) dentro del contenido de Flash Player en el navegador. Seleccione la ficha Visualización, que es la primera ficha, y active la casilla de verificación: Activar aceleración de hardware.

Modos de ventana directo y de composición con GPU

Flash Player 10 introduce dos modos de ventana, directo y composición por GPU, que se pueden habilitar desde la configuración de publicación de la herramienta de edición de Flash. Estos modos no se admiten en AIR. Para poder utilizar estos modos, es preciso activar la aceleración de hardware para Flash Player.

El modo directo utiliza la ruta más rápida y directa para introducir gráficos en pantalla, lo que resulta apropiado para la reproducción de vídeo.

La composición con GPU utiliza la unidad de procesamiento de gráficos de la tarjeta de vídeo para acelerar la composición. La composición de vídeo es el proceso de organización en capas de varias imágenes para crear una sola imagen de vídeo. Cuando la composición se acelera con la GPU, es posible mejorar el rendimiento de la conversión YUV, la corrección de color, la rotación o la escala, así como la fusión. La conversión YUV hace referencia a la conversión de color de las señales analógicas del compuesto, que se utilizan para la transmisión, en el modelo de color RGB (rojo, verde, azul) que utilizan las pantallas y las cámaras de vídeo. El uso de la GPU para acelerar la composición reduce la demanda del equipo y la memoria que se aplica a la CPU. También supone una reproducción más suavizada para el vídeo de definición estándar.

Sea cuidadoso al implementar estos modos de ventana. El uso de la composición con GPU puede resultar costoso en términos de recursos de CPU y memoria. Si algunas operaciones (por ejemplo, modos de fusión, recorte o enmascaramiento) no se pueden llevar a cabo en la GPU, se realizan mediante el software. Adobe recomienda la limitación a un archivo SWF por página HTML cuando se utilicen estos modos, que no deben se habilitar para banners. La función Probar película de Flash no utiliza la aceleración de hardware, pero puede emplearla mediante la opción Vista previa de publicación.

La definición de una velocidad de fotogramas en el archivo SWF que sea superior a 60, velocidad máxima de actualización de pantalla, no resulta útil. El establecimiento de la velocidad de fotogramas de 50 a 55 permite fotogramas eliminados, lo cual puede suceder por distintos motivos en algunas ocasiones.

Para poder utilizar el modo directo, es preciso disponer de Microsoft DirectX 9 con VRAM de 128 MB en Windows y OpenGL para Apple Macintosh, Mac OS X v10.2 o versiones posteriores. El modo de composición por GPU requiere compatibilidad con Microsoft DirectX 9 y Pixel Shader 2.0 en Windows con 128 MB de VRAM. En Mac OS X y Linux, la composición por GPU requiere OpenGL 1.5 y diversas extensiones de OpenGL (objeto framebuffer, objetos multitexture y shader, lenguaje de sombreado y sombreados de fragmentos).

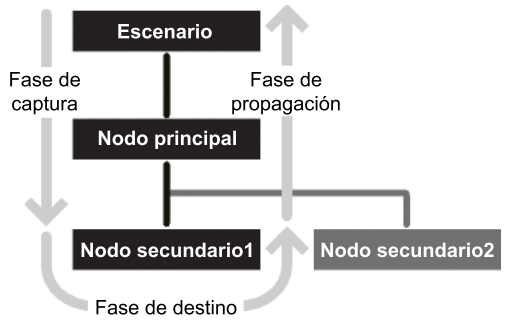
Puede activar los modos de aceleración `directo` y `gpu` por archivo SWF mediante el cuadro de diálogo de configuración de publicación de Flash, utilizando el menú de aceleración de hardware de la ficha Flash. Si selecciona Ninguno, el modo de ventana vuelve a los valores `predeterminado`, `transparente` u `opaco`, tal y como se especifica en la configuración del modo de ventana en la ficha HTML.

Gestión de eventos de objetos de visualización

La clase DisplayObject hereda de la clase EventDispatcher. Esto significa que cada uno de los objetos de visualización puede participar completamente en el modelo de eventos (descrito en "Gestión de eventos" en la página 254). Cada objeto de visualización puede utilizar su método `addEventListener()`, heredado de la clase EventDispatcher, para detectar un evento concreto, pero solamente si el objeto detector forma parte del flujo de dicho evento.

Cuando Flash Player o AIR distribuyen un objeto de evento, éste realiza un viaje de ida y vuelta del escenario al objeto de visualización donde se produjo el evento. Por ejemplo, si un usuario hace clic en un objeto de visualización denominado `child1`, Flash Player distribuye un objeto de evento desde el escenario, y a través de la jerarquía de la lista de visualización, hasta el objeto de visualización `child1`.

El flujo del evento se divide conceptualmente en tres fases, tal y como se ilustra en el diagrama siguiente:



Para obtener más información, consulte "Gestión de eventos" en la página 254.

Un aspecto importante que hay que tener en cuenta cuando se trabaja con eventos de objetos de visualización es el efecto que los detectores de eventos pueden tener sobre la eliminación automática de objetos de visualización de la memoria (recopilados como datos innecesarios) cuando se quitan de la lista de visualización. Si un objeto de visualización tiene objetos suscritos a sus eventos como detectores, dicho objeto de visualización no se eliminará de la memoria aunque se quite de la lista de visualización, porque seguirá teniendo referencias a esos objetos detectores. Para obtener más información, consulte "Administración de detectores de eventos" en la página 268.

Selección de una subclase DisplayObject

Una de las decisiones importantes que hay que tomar cuando se trabaja con objetos de visualización es elegir qué objeto de visualización se va a utilizar para cada propósito. A continuación se ofrecen algunas directrices que pueden ser útiles en la toma de esta decisión. Estas mismas sugerencias se aplican tanto si se necesita una instancia de una clase como si se elige una clase base para una clase que se esté creando:

- Si no se necesita un objeto que pueda ser un contenedor de otros objetos de visualización (es decir, que sólo se necesita uno que actúe como un elemento de pantalla autónomo), debe elegirse una de estas subclases DisplayObject o InteractiveObject, dependiendo del uso que se le vaya a dar:
  - Bitmap para mostrar una imagen de mapa de bits.
  - TextField para añadir texto.
  - Video para mostrar vídeo.
  - Shape para obtener un "lienzo" donde dibujar contenido en pantalla. En concreto, si se desea crear una instancia para dibujar formas en la pantalla y no se va a utilizar como contenedor de otros objetos de visualización, se obtendrán importantes ventajas de rendimiento si se usa Shape en lugar de Sprite o MovieClip.
  - MorphShape, StaticText o SimpleButton para los elementos creados con la herramienta de edición de Flash. (No es posible crear instancias de estas clases mediante programación, pero sí crear variables con estos tipos de datos para hacer referencia a los elementos creados con la herramienta de edición de Flash.)
- Si se necesita una variable para hacer referencia al escenario principal, debe usarse la clase Stage como su tipo de datos.
- Si se necesita un contenedor para cargar un archivo SWF o un archivo de imagen externo, se debe usar una instancia de Loader. El contenido cargado se añadirá a la lista de visualización como un elemento secundario de la instancia de Loader. Su tipo de datos dependerá de la naturaleza del contenido cargado, como se indica a continuación:
  - Una imagen cargada será una instancia de Bitmap.
  - Un archivo SWF cargado, escrito en ActionScript 3.0, será una instancia de Sprite o MovieClip (o una instancia de una subclase de esas clases, según lo especifique el creador de contenido).
  - Un archivo SWF cargado, escrito en ActionScript 1.0 o ActionScript 2.0, será una instancia de AVM1Movie.
- Si se necesita que un objeto actúe como contenedor de otros objetos de visualización, aunque se vaya a dibujar o no en el objeto de visualización mediante ActionScript, debe elegirse una de las subclases de DisplayObjectContainer:
  - Sprite si el objeto se creará solamente con ActionScript, o como la clase base de un objeto de visualización personalizado que se creará y manipulará exclusivamente con ActionScript.
  - MovieClip si se crea una variable para hacer referencia a un símbolo de clip de película creado en la herramienta de edición de Flash.
- Si se crea una clase que se asociará a un símbolo de clip de película en la biblioteca de Flash, se debe elegir una de las siguientes subclases de DisplayObjectContainer como clase base:
  - MovieClip si el símbolo de clip de película asociado tiene contenido en más de un fotograma.
  - Sprite si el símbolo de clip de película asociado tiene contenido sólo en el primer fotograma.

Manipulación de objetos de visualización

Independientemente del objeto de visualización que se decida utilizar, hay una serie de manipulaciones comunes a todos los objetos de visualización como elementos que se muestran en la pantalla. Por ejemplo, todos ellos pueden colocarse en la pantalla, moverse hacia adelante o hacia atrás en el orden de apilamiento de los objetos de visualización, cambiar de escala, girarse, etc. Dado que todos los objetos de visualización heredan esta funcionalidad de su clase base común (DisplayObject), esta funcionalidad tiene el mismo comportamiento si se manipula una instancia de TextField, una instancia de Video, una instancia de Shape o cualquier otro objeto de visualización. En las secciones siguientes se detallan algunas de estas manipulaciones comunes a los objetos de visualización.

Cambio de posición

La manipulación más básica de cualquier objeto de visualización es su colocación en la pantalla. Para definir la posición de un objeto de visualización, deben cambiarse las propiedades `x` e `y` del objeto.

```
myShape.x = 17;
myShape.y = 212;
```

El sistema de colocación de objetos de visualización trata el escenario como un sistema de coordenadas cartesiano (el sistema de cuadrícula habitual, con un eje x horizontal y un eje y vertical). El origen del sistema de coordenadas (la coordenada 0,0 donde convergen los ejes x e y) está situado en la esquina superior izquierda del escenario. Tomando esta coordenada como punto de referencia, los valores x son positivos si están a la derecha y negativos si están a la izquierda, mientras que los valores y son positivos si están abajo y negativos si están arriba (al contrario que en los sistemas gráficos típicos). Por ejemplo, las anteriores líneas de código mueven el objeto `myShape` a la coordenada x 17 (17 píxeles a la derecha del origen) y a la coordenada y 212 (212 píxeles por debajo del origen).

De forma predeterminada, cuando se crea un objeto de visualización con ActionScript, las propiedades `x` e `y` se establecen en 0, colocando el objeto en la esquina superior izquierda de su contenido principal.

Cambio de posición con respecto al escenario

Es importante recordar que las propiedades `x` e `y` siempre hacen referencia a la posición del objeto de visualización con respecto a la coordenada 0,0 de los ejes de su objeto de visualización principal. De este modo, en una instancia de Shape (por ejemplo, un círculo) contenida en una instancia de Sprite, al establecer en 0 las propiedades `x` e `y` del objeto Shape, se colocará un círculo en la esquina superior izquierda del objeto Sprite, que no es necesariamente la esquina superior izquierda del escenario. Para colocar un objeto con respecto a las coordenadas globales del escenario, se puede usar el método `globalToLocal()` de cualquier objeto de visualización para convertir las coordenadas globales (escenario) en coordenadas locales (contenedor de objeto de visualización), como en el ejemplo siguiente:

```
// Position the shape at the top-left corner of the Stage,
// regardless of where its parent is located.

// Create a Sprite, positioned at x:200 and y:200.
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// Draw a dot at the Sprite's 0,0 coordinate, for reference.
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();

// Create the circle Shape instance.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Draw a circle with radius 50 and center point at x:50, y:50 in the Shape.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Move the Shape so its top-left corner is at the Stage's 0, 0 coordinate.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

También se puede usar el método `localToGlobal()` de la clase DisplayObject para convertir las coordenadas locales en coordenadas del escenario.

Creación de interacción de arrastrar y colocar

Una de las razones habituales para mover un objeto de visualización es crear una interacción de arrastrar y colocar, de forma que cuando el usuario haga clic en un objeto, éste se mueva con el movimiento del ratón, hasta que se suelte el botón del ratón. Hay dos maneras posibles de crear una interacción de arrastrar y colocar en ActionScript. En cualquier caso, se utilizan dos eventos de ratón: cuando se presiona el botón del ratón el objeto recibe la orden de seguir al cursor del ratón y, cuando se suelta el botón, recibe la orden de dejar de seguirlo.

La primera opción consiste en usar el método `startDrag()` y es más sencilla pero también más limitada. Cuando se presiona el botón del ratón, se llama al método `startDrag()` del objeto de visualización que se va a arrastrar. Cuando se suelta el botón del ratón, se llama el método `stopDrag()`.

```
// This code creates a drag-and-drop interaction using the startDrag()
// technique.
// square is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    square.startDrag();
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Esta técnica presenta una limitación realmente importante: con `startDrag()` únicamente se puede arrastrar un elemento cada vez. Si se arrastra un objeto de visualización y se llama al método `startDrag()` en otro objeto de visualización, el primer objeto deja de seguir al ratón inmediatamente. Por ejemplo, si se cambia la función `startDragging()` como se muestra aquí, sólo se arrastrará el objeto `circle`, en lugar de la llamada al método `square.startDrag()`:

```
function startDragging(event:MouseEvent):void
{
    square.startDrag();
    circle.startDrag();
}
```

Como consecuencia de poder arrastrar un solo objeto cada vez cuando se usa `startDrag()`, es posible llamar al método `stopDrag()` en cualquier objeto de visualización y detenerlo sea cual sea el objeto que se esté arrastrando.

Si se necesita arrastrar más de un objeto de visualización o evitar la posibilidad de conflictos porque más de un objeto pueda utilizar `startDrag()`, se recomienda usar la técnica de seguimiento del ratón para crear el efecto de arrastre. Con esta técnica, cuando se presiona el botón del ratón, se suscribe una función al evento `mouseMove` del escenario, como detector. Esta función, que se llama cada vez que se mueve el ratón, hace que el objeto arrastrado salte a las coordenadas x, y del ratón. Cuando se suelta el botón del ratón, se quita la suscripción de la función como detector, de modo que ya no se llama cuando se mueve el ratón y el objeto deja de seguir al cursor del ratón. En el código siguiente se muestra esta técnica:

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // Record the difference (offset) between where
    // the cursor was when the mouse button was pressed and the x, y
    // coordinate of the circle when the mouse button was pressed.
    offsetX = event.stageX - circle.x;
    offsetY = event.stageY - circle.y;

    // tell Flash Player to start listening for the mouseMove event
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragCircle(event:MouseEvent):void
{
    // Move the circle to the location of the cursor, maintaining
    // the offset between the cursor's location and the
    // location of the dragged object.
    circle.x = event.stageX - offsetX;
    circle.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Además de hacer que un objeto de visualización siga al cursor del ratón, una parte común de la interacción de arrastrar y colocar consiste en mover el objeto arrastrado al frente de la pantalla, de modo que parezca que flota sobre todos los demás objetos. Por ejemplo, supongamos que se tienen dos objetos, un círculo y un cuadrado, y que ambos tienen una interacción de arrastrar y colocar. Si el círculo está por debajo del cuadrado en la lista de visualización, y se hace clic en el círculo y se arrastra de modo que el cursor quede sobre el cuadrado, el círculo parecerá deslizarse detrás del cuadrado, rompiendo así la ilusión de arrastrar y colocar. En lugar de eso, se puede hacer que el círculo se mueva a la parte superior de la lista de visualización cuando se hace clic en él, de modo que el círculo siempre aparezca encima de cualquier otro contenido.

El código siguiente (adaptado del ejemplo anterior) crea una interacción de arrastrar y colocar para dos objetos de visualización: un círculo y un cuadrado. Cuando se presiona el botón del ratón sobre cualquiera de los dos, el elemento en cuestión se mueve a la parte superior de la lista de visualización del escenario, de modo que el elemento arrastrado aparece siempre encima. El código nuevo o cambiado con respecto a la lista anterior aparece en negrita.

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle and square are DisplayObjects (e.g. MovieClip or Sprite
// instances).

import flash.display.DisplayObject;
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // remember which object is being dragged
    draggedObject = DisplayObject(event.target);

    // Record the difference (offset) between where the cursor was when
    // the mouse button was pressed and the x, y coordinate of the
    // dragged object when the mouse button was pressed.
    offsetX = event.stageX - draggedObject.x;
    offsetY = event.stageY - draggedObject.y;

    // move the selected object to the top of the display list
    stage.addChild(draggedObject);

    // Tell Flash Player to start listening for the mouseMove event.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}
```

```
// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragObject(event:MouseEvent):void
{
    // Move the dragged object to the location of the cursor, maintaining
    // the offset between the cursor's location and the location
    // of the dragged object.
    draggedObject.x = event.stageX - offsetX;
    draggedObject.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Para aumentar este efecto, por ejemplo en un juego en el que se mueven fichas o cartas entre distintas pilas, se podría añadir el objeto arrastrado a la lista de visualización del escenario cuando se selecciona y luego añadirlo a otra lista de visualización (por ejemplo, la pila donde se coloca), cuando se suelta el botón del ratón.

Finalmente, para mejorar el efecto, se podría aplicar un filtro de sombra al objeto de visualización al hacer clic en él (al iniciar el arrastre) y quitar la sombra al soltar el objeto. Para obtener información detallada sobre el uso del filtro de sombra y otros filtros de objetos de visualización en ActionScript, consulte "Aplicación de filtros a objetos de visualización" en la página 363.

Desplazamiento lateral y vertical de objetos de visualización

Si un objeto de visualización es demasiado grande para el área donde se desea mostrar, se puede utilizar la propiedad `scrollRect` para definir el área visualizable del objeto de visualización. Además, si se cambia la propiedad `scrollRect` como respuesta a la entrada del usuario, se puede desplazar el contenido horizontal o verticalmente.

La propiedad `scrollRect` es una instancia de la clase Rectangle, que es una clase que combina los valores necesarios para definir un área rectangular como un solo objeto. Para definir inicialmente el área visualizable del objeto de visualización, se debe crear una nueva instancia de Rectangle y asignarla a la propiedad `scrollRect` del objeto de visualización. Posteriormente, para el desplazamiento vertical u horizontal, se lee la propiedad `scrollRect` en una variable Rectangle independiente y se cambia la propiedad deseada (por ejemplo, la propiedad `x` de la instancia de Rectangle para el desplazamiento horizontal o la propiedad `y` para el desplazamiento vertical). A continuación, se reasigna dicha instancia de Rectangle a la propiedad `scrollRect` para notificar el valor cambiado al objeto de visualización.

Por ejemplo, el código siguiente define el área visualizable de un objeto TextField denominado `bigText` que es demasiado alto para los límites del archivo SWF. Cuando se hace clic en dos botones denominados `up` y `down`, se llama a funciones que desplazan verticalmente el contenido del objeto TextField a través de la modificación de la propiedad `y` de la instancia de Rectangle `scrollRect`.

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// Define the initial viewable area of the TextField instance:
// left: 0, top: 0, width: TextField's width, height: 350 pixels.
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// Cache the TextField as a bitmap to improve performance.
bigText.cacheAsBitmap = true;

// called when the "up" button is clicked
function scrollUp(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Decrease the y value of the rectangle by 20, effectively
    // shifting the rectangle down by 20 pixels.
    rect.y -= 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

// called when the "down" button is clicked
function scrollDown(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Increase the y value of the rectangle by 20, effectively
    // shifting the rectangle up by 20 pixels.
    rect.y += 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

Tal y como se ilustra en este ejemplo, cuando se trabaja con la propiedad `scrollRect` de un objeto de visualización, es mejor especificar que Flash Player o AIR deben almacenar en caché el contenido del objeto de visualización como un mapa de bits, usando para ello la propiedad `cacheAsBitmap`. De este modo, Flash Player o AIR no tienen que volver a dibujar todo el contenido del objeto de visualización cada vez que se desplaza verticalmente y, en su lugar, pueden utilizar el mapa de bits almacenado en caché para representar la parte necesaria directamente en pantalla. Para obtener más información, consulte "Almacenamiento en caché de los objetos de visualización" en la página 309.

Manipulación del tamaño y ajuste de escala de los objetos

Hay dos formas posibles de medir y manipular el tamaño de un objeto de visualización: con las propiedades de dimensión (`width` y `height`) o las propiedades de escala (`scaleX` y `scaleY`).

Cada objeto de visualización tiene una propiedad `width` y una propiedad `height`, definidas inicialmente con el tamaño del objeto en píxeles. La lectura de los valores de estas propiedades permite medir el tamaño del objeto de visualización. También es posible especificar nuevos valores para cambiar el tamaño del objeto, como se indica a continuación:

```
// Resize a display object.
square.width = 420;
square.height = 420;

// Determine the radius of a circle display object.
var radius:Number = circle.width / 2;
```

Al cambiar la altura (`height`) o anchura (`width`) de un objeto de visualización, se ajusta la escala del objeto, de modo que el contenido del mismo se contrae o expande para ajustarse a la nueva área. Si el objeto de visualización sólo contiene formas vectoriales, dichas formas volverán a dibujarse en la nueva escala sin perder la calidad. Los elementos gráficos de mapa de bits del objeto de visualización ajustarán su escala en lugar de dibujarse de nuevo. Así pues, por ejemplo, una foto digital cuya anchura y altura hayan aumentado más allá de las dimensiones reales de la información de píxeles de la imagen aparecerá pixelada y con bordes dentados.

Cuando se cambian las propiedades `width` o `height` de un objeto de visualización, Flash Player y AIR actualizan también las propiedades `scaleX` y `scaleY` del objeto.

Nota: *los objetos TextField son una excepción a este comportamiento de escala. Los campos de texto deben ajustar su tamaño automáticamente para alojar el texto y los tamaños de fuente. Por ello, reajustan sus valores scaleX o scaleY a 1 tras el cambio de tamaño. Sin embargo, si se ajustan los valores scaleX o scaleY de un objeto TextField, los valores de anchura y altura cambian para escalar los valores proporcionados.*

Estas propiedades representan el tamaño relativo del objeto de visualización con respecto a su tamaño original. Las propiedades `scaleX` y `scaleY` utilizan valores fraccionarios (decimales) para representar el porcentaje. Por ejemplo, si se cambia la anchura (`width`) de un objeto de visualización a la mitad de su tamaño original, la propiedad `scaleX` del objeto tendrá el valor `.5`, que significa 50 por ciento. Si se duplica la altura, la propiedad `scaleY` tendrá el valor `2`, que significa 200 por ciento.

```
// circle is a display object whose width and height are 150 pixels.
// At original size, scaleX and scaleY are 1 (100%).
trace(circle.scaleX); // output: 1
trace(circle.scaleY); // output: 1

// When you change the width and height properties,
// Flash Player changes the scaleX and scaleY properties accordingly.
circle.width = 100;
circle.height = 75;
trace(circle.scaleX); // output: 0.6622516556291391
trace(circle.scaleY); // output: 0.4966887417218543
```

Los cambios de tamaño no son proporcionales. Dicho de otro modo, si se cambia la altura (`height`) de un cuadrado pero no la anchura (`width`), ya no tendrá las mismas proporciones y ya no será un cuadrado sino un rectángulo. Si se desea cambiar relativamente el tamaño de un objeto de visualización, se pueden definir los valores de las propiedades `scaleX` y `scaleY` para cambiar el tamaño del objeto, en lugar de definir las propiedades `width` o `height`. Por ejemplo, este código cambia la anchura (`width`) del objeto de visualización denominado `square` y luego modifica la escala vertical (`scaleY`) para que coincida con la escala horizontal, de modo que el tamaño del cuadrado siga siendo proporcional.

```
// Change the width directly.
square.width = 150;

// Change the vertical scale to match the horizontal scale,
// to keep the size proportional.
square.scaleY = square.scaleX;
```

Control de la distorsión durante el ajuste de escala

Normalmente, cuando se ajusta la escala de un objeto de visualización (por ejemplo, si se expande horizontalmente), la distorsión resultante se extiende por igual en todo el objeto, de modo que cada parte se expande en la misma cantidad. Este es probablemente el comportamiento deseado para los gráficos y elementos de diseño. Sin embargo, a veces es preferible tener el control de las partes del objeto de visualización que se expanden y de las que permanecen invariables. Un ejemplo habitual es un botón que es un rectángulo con bordes redondeados. Con el ajuste de escala normal, las esquinas del botón se expandirán de modo que el radio de la esquina cambie a medida que el botón cambia de tamaño.

Sin embargo, en este caso es preferible tener el control del ajuste de escala, es decir, poder designar las áreas en las que debe ajustarse la escala (los lados rectos y la parte central) y las áreas en las que no, de forma que el ajuste de escala se produzca sin una distorsión visible.

Se puede usar la escala en 9 divisiones (Scale-9) para crear objetos de visualización cuyo ajuste de escala pueda controlarse. Con la escala en 9 divisiones, el objeto de visualización se divide en nueve rectángulos independientes (una cuadrícula de 3 por 3, como la cuadrícula de un tablero de tres en raya). Los rectángulos no son necesariamente del mismo tamaño; el usuario designa la ubicación de las líneas de la cuadrícula. El contenido de los cuatro rectángulos de bordes redondeados (como las esquinas redondeadas de un botón) no se expandirá ni comprimirá cuando se ajuste la escala del objeto de visualización. Los rectángulos superior central e inferior central ajustarán la escala horizontalmente pero no verticalmente, mientras que los rectángulos izquierdo central y derecho central ajustarán la escala verticalmente pero no horizontalmente. El rectángulo central ajustará la escala tanto vertical como horizontalmente.

Teniendo esto en cuenta, cuando se crea un objeto de visualización y se desea que nunca se ajuste la escala de determinado contenido, simplemente hay que asegurarse de que las líneas divisorias de la cuadrícula de escala en 9 divisiones se sitúen de forma que el contenido finalice en uno de los rectángulos de bordes redondeados.

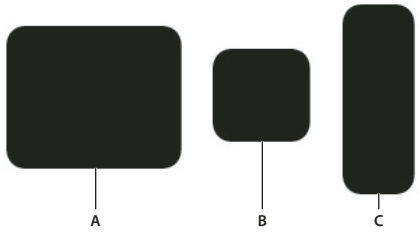
En ActionScript, al establecer un valor de la propiedad `scale9Grid` de un objeto de visualización, se activa la escala en 9 divisiones en el objeto y se define el tamaño de los rectángulos en la cuadrícula Scale-9 del objeto. Se utiliza una instancia de la clase Rectangle como valor de la propiedad `scale9Grid`, del siguiente modo:

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

Los cuatro parámetros del constructor de Rectangle son la coordenada x, la coordenada y, la anchura y la altura. En este ejemplo, la esquina superior izquierda del rectángulo se sitúa en el punto x: 32, y: 27 en el objeto de visualización denominado `myButton`. El rectángulo tiene una anchura de 71 píxeles y una altura de 64 píxeles (es decir, su borde derecho está en la coordenada x 103 del objeto de visualización y su borde inferior en la coordenada y 92 del objeto de visualización).

El área real contenida en el área definida por la instancia de Rectangle representa el rectángulo central de la cuadrícula Scale-9. Para calcular los demás rectángulos, Flash Player y AIR amplían los lados de la instancia de Rectangle, como se muestra a continuación:

En este caso, a medida que el botón ajusta la escala para expandirse o comprimirse, las esquinas redondeadas no se expanden ni comprimen, pero las demás áreas se ajustan a la escala.



A. *myButton.width = 131;myButton.height = 106;* ***B.*** *myButton.width = 73;myButton.height = 69;* ***C.*** *myButton.width = 54;myButton.height = 141;*

Almacenamiento en caché de los objetos de visualización

El tamaño de sus diseños en Flash irá creciendo, tanto si está creando una aplicación como si realiza complejas animaciones mediante scripts, por lo que deberá tener en cuenta el rendimiento y la optimización. Si tiene contenido que permanece estático (como una instancia de Shape rectangular), Flash Player y AIR no optimizarán el contenido. Por lo tanto, si cambia la posición del rectángulo, Flash Player o AIR redibujan toda la instancia de Shape.

Se pueden almacenar en caché los objetos de visualización especificados para mejorar el rendimiento del archivo SWF. El objeto de visualización es una *superficie*, básicamente una versión de mapa de bits de los datos vectoriales de la instancia, que son datos que no deseará que cambien mucho a lo largo del archivo SWF. Por consiguiente, las instancias para las que está activada la caché no se vuelven a dibujar continuamente mientras se reproduce el archivo SWF, lo que permite que el archivo SWF se represente rápidamente.

Nota: *es posible actualizar los datos vectoriales, momento en el cual se recrea la superficie. Por tanto, los datos vectoriales almacenados en caché en la superficie no tienen por qué permanecer intactos en todo el archivo SWF.*

Al establecer la propiedad `cacheAsBitmap` del objeto de visualización en `true`, el objeto de visualización almacena en caché una representación de mapa de bits de sí mismo. Flash Player o AIR crean un objeto de superficie para la instancia, que es un mapa de bits almacenado en caché, en lugar de los datos vectoriales. Si cambia los límites del objeto de visualización, la superficie se recrea en lugar de modificarse su tamaño. Las superficies pueden anidarse dentro de otras superficies. La superficie secundaria copia el mapa de bits en su superficie principal. Para obtener más información, consulte "Activación de almacenamiento de mapas de bits en caché" en la página 311.

Las propiedades `opaqueBackground` y `scrollRect` de la clase DisplayObject se relacionan con el almacenamiento en caché de mapa de bits a través de la propiedad `cacheAsBitmap`. Aunque estas propiedades son independientes entre sí, las propiedades `opaqueBackground` y `scrollRect` funcionan mejor cuando un objeto se almacena en caché como un mapa de bits; las ventajas de rendimiento de las propiedades `opaqueBackground` y `scrollRect` sólo se evidencian cuando se establece `cacheAsBitmap` en `true`. Para obtener más información sobre el desplazamiento del contenido de objetos de visualización, consulte "Desplazamiento lateral y vertical de objetos de visualización" en la página 305. Para obtener más información sobre la configuración de un fondo opaco, consulte "Configuración de un color de fondo opaco" en la página 312.

Para obtener información sobre el enmascaramiento del canal alfa, que requiere que establezca la propiedad `cacheAsBitmap` como `true`, consulte "Enmascarar objetos de visualización" en la página 316.

Cuándo es conveniente activar la caché

La activación de la caché para un objeto de visualización crea una superficie, lo que presenta algunas ventajas como, por ejemplo, una mayor velocidad de representación de animaciones vectoriales complejas. Existen varias situaciones en las que deseará activar la caché. Podría parecer que siempre es preferible activar la caché para mejorar el rendimiento de los archivos SWF; sin embargo, hay situaciones en las que la activación de la caché no mejora el rendimiento e incluso lo reduce. En esta sección se describen situaciones en las que debe utilizarse la activación de la caché y en las que deben emplearse objetos de visualización normales.

El rendimiento global de los datos almacenados en caché depende de la complejidad de los datos vectoriales de las instancias, de la cantidad de datos que cambie y de si ha establecido la propiedad `opaqueBackground`. Si cambia zonas pequeñas, la diferencia entre el uso de una superficie y el uso de datos vectoriales puede ser insignificante. Es aconsejable probar ambas situaciones antes de implementar la aplicación.

Cuándo es conveniente utilizar la caché de mapa de bits

A continuación se incluyen situaciones típicas en las que pueden apreciarse ventajas significativas al activar la caché de mapa de bits.

- Imagen de fondo compleja Una aplicación que contiene una imagen de fondo compleja y detallada de datos de vectoriales (quizás una imagen en la que aplica el comando Trazar mapa de bits o ilustraciones que ha creado en Adobe Illustrator®). Podrían animarse los caracteres del fondo, lo que ralentizaría la animación porque el fondo necesita la regeneración constante de los datos vectoriales. Para mejorar el rendimiento, se puede establecer la propiedad `opaqueBackground` del objeto de visualización de fondo en `true`. El fondo se representa como mapa de bits y puede redibujarse rápidamente, por lo que la animación se reproduce con mucha mayor velocidad.
- Campo de texto con desplazamiento: aplicación que muestra una gran cantidad de texto en un campo de texto con desplazamiento. Se puede colocar el campo de texto en un objeto de visualización definido como desplazable con límites con desplazamiento (propiedad `scrollRect`). De este modo, se permite un desplazamiento rápido por los píxeles de la instancia especificada. Cuando el usuario se desplaza por la instancia del objeto de visualización, Flash Player o AIR mueven hacia arriba los píxeles desplazados y generan la región recién expuesta en lugar de regenerar todo el campo de texto.

- Sistema de ventanas: aplicación con un complejo sistema de ventanas superpuestas. Cada ventana puede abrirse o cerrarse (por ejemplo, las ventanas de un navegador Web). Si se marca cada ventana como una superficie (estableciendo la propiedad `cacheAsBitmap` en `true`), cada ventana se aísla y se almacena en caché. Los usuarios pueden arrastrar las ventanas para que se puedan superponer y cada ventana no necesita regenerar el contenido vectorial.
- Enmascaramiento del canal alfa: si se usa el enmascaramiento del canal alfa, la propiedad `cacheAsBitmap` se debe establecer en `true`. Para obtener más información, consulte "Enmascarar objetos de visualización" en la página 316.

La activación de la caché de mapa de bits en todas estas situaciones mejora el nivel de respuesta e interactividad de la aplicación, pues optimiza los gráficos vectoriales.

Asimismo, si se aplica un filtro a un objeto de visualización, `cacheAsBitmap` se establece automáticamente en `true`, aunque el usuario lo establezca explícitamente en `false`. Si se quitan todos los filtros del objeto de visualización, la propiedad `cacheAsBitmap` recupera el valor que se estableció por última vez.

Cuándo es conveniente evitar utilizar la caché de mapa de bits

La utilización inadecuada de esta función puede afectar negativamente al archivo SWF. Cuando utilice la caché de mapa de bits, recuerde las siguientes directrices:

- No haga un uso abusivo de las superficies (objetos de visualización para los que está activada la caché). Cada superficie utiliza más memoria que un objeto de visualización normal, lo que significa que sólo deberá utilizar las superficies cuando necesite mejorar el rendimiento de la representación.

  Un mapa de bits almacenado en caché utiliza bastante más memoria que un objeto de visualización normal. Por ejemplo, si una instancia de Sprite en el escenario tiene un tamaño de 250 por 250 píxeles, al almacenarse en caché podría utilizar 250 KB en lugar de 1 KB cuando se trata de una instancia de Sprite normal (no almacenada en caché).
- Evite aplicar zoom a las superficies almacenadas en caché. Si utiliza en exceso la caché de mapa de bits, se consume una gran cantidad de memoria (consulte el apartado anterior), especialmente si aplica el zoom para acercar el contenido.
- Utilice superficies para instancias de objetos de visualización que sean principalmente estáticas (sin animación). Puede arrastrar o mover la instancia, pero el contenido de la instancia no debe incluir demasiada animación ni cambiar mucho. (La animación o el cambio de contenido son más frecuentes en las instancias de MovieClip que contienen animación o en las instancias de Video.) Por ejemplo, si gira o transforma una instancia, ésta cambia entre la superficie y los datos vectoriales, lo que dificulta el procesamiento y afecta de forma negativa al archivo SWF.
- Si mezcla superficies con datos vectoriales, aumentará la cantidad de procesamiento que deben llevar a cabo Flash Player y AIR (y, a veces, el equipo). Agrupe las superficies tanto como sea posible; por ejemplo, cuando cree aplicaciones de gestión de ventanas.

Activación de almacenamiento de mapas de bits en caché

Para activar la caché de mapa de bits para un objeto de visualización, debe establecerse su propiedad `cacheAsBitmap` en `true`:

```
mySprite.cacheAsBitmap = true;
```

Una vez establecida la propiedad `cacheAsBitmap` en `true`, es posible observar que el objeto de visualización ajusta automáticamente los píxeles a coordenadas enteras. Al probar el archivo SWF, debería apreciarse un aumento considerable en la velocidad de representación de imágenes vectoriales complejas.

Si se dan una o varias de las siguientes condiciones, no se crea ninguna superficie (mapa de bits almacenado en caché) aunque `cacheAsBitmap` se haya establecido en `true`:

- El mapa de bits tiene una altura o una anchura superior a 2880 píxeles.
- No se puede asignar el mapa de bits (se produce un error de memoria insuficiente).

Configuración de un color de fondo opaco

Se puede configurar un fondo opaco en un objeto de visualización. Por ejemplo, cuando el archivo SWF tiene un fondo que contiene complejos gráficos vectoriales, se puede establecer la propiedad `opaqueBackground` en un color especificado (normalmente el mismo color del escenario). El color se especifica en forma de número (normalmente un valor de color hexadecimal). El fondo se trata entonces como un mapa de bits, lo que ayuda a optimizar el rendimiento.

Cuando se establece la propiedad `cacheAsBitmap` en `true` y la propiedad `opaqueBackground` en un color especificado, la propiedad `opaqueBackground` permite que el mapa de bits interno sea opaco y se represente más rápido. Si no se establece `cacheAsBitmap` en `true`, la propiedad `opaqueBackground` añade una forma cuadrada vectorial opaca al fondo del objeto de visualización. No crea un mapa de bits automáticamente.

En el ejemplo siguiente se muestra cómo configurar el fondo de un objeto de visualización para optimizar el rendimiento:

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

En este caso, el color de fondo del objeto Shape denominado `myShape` se establece en rojo (`0xFF0000`). Suponiendo que la instancia de Shape contiene un dibujo de un triángulo verde, en un escenario con un fondo blanco, se mostraría como un triángulo verde con color rojo en el espacio vacío del recuadro de delimitación de la instancia de Shape (el rectángulo que abarca completamente Shape).

Por supuesto, este código tendría más sentido si se utilizara con un escenario que tuviera un fondo de color rojo sólido. Con un fondo de otro color, dicho color se especificaría en su lugar. Por ejemplo, en un archivo SWF con un fondo blanco, la propiedad `opaqueBackground` probablemente se establecería en `0xFFFFFF`, o blanco puro.

Aplicación de modos de mezcla

Los modos de mezcla implican combinar los colores de una imagen (la imagen base) con los de otra imagen (la imagen de mezcla) para producir una tercera imagen, que es la que se muestra realmente en la pantalla. Cada valor de píxel de una imagen se procesa con el correspondiente valor de píxel de la otra imagen para producir un valor de píxel para esa misma posición en el resultado.

Cada objeto de visualización tiene una propiedad `blendMode` que puede establecerse en uno de los siguientes modos de mezcla. Se trata de constantes definidas en la clase BlendMode. Como alternativa, se pueden usar los valores de String (entre paréntesis) que son los valores reales de las constantes.

- `BlendMode.ADD` (`"add"`): suele utilizarse para crear un efecto de disolución de aclarado animado entre dos imágenes.
- `BlendMode.ALPHA` (`"alpha"`): suele utilizarse para aplicar la transparencia del primer plano al fondo.

- `BlendMode.DARKEN` (`"darken"`): suele utilizarse para superponer el tipo.
- `BlendMode.DIFFERENCE` (`"difference"`): suele utilizarse para crear colores más vivos.
- `BlendMode.ERASE` (`"erase"`): se suele utilizar para recortar (borrar) parte del fondo utilizando el alfa de primer plano.
- `BlendMode.HARDLIGHT` (`"hardlight"`): suele utilizase para crear efectos de sombreado.
- `BlendMode.INVERT` (`"invert"`): se utiliza para invertir el fondo.
- `BlendMode.LAYER` (`"layer"`): se utiliza para forzar la creación de un búfer temporal para la composición previa de un determinado objeto de visualización.
- `BlendMode.LIGHTEN` (`"lighten"`): suele utilizarse para superponer el tipo.
- `BlendMode.MULTIPLY` (`"multiply"`): suele utilizarse para crear efectos de sombras y profundidad.
- `BlendMode.NORMAL` (`"normal"`): se utiliza para especificar que los valores de píxeles de la imagen mezclada sustituyan a los de la imagen base.
- `BlendMode.OVERLAY` (`"overlay"`): se suele utilizar para crear efectos de sombreado.
- `BlendMode.SCREEN` (`"screen"`): se suele emplear para crear resaltados y destellos de lentes.
- `BlendMode.SHADER` (`"shader"`): se utiliza para especificar que se usa un sombreado de Pixel Bender para crear un efecto de fusión personalizado. Para obtener más información sobre el uso de sombreados, consulte "Trabajo con sombreados de Pixel Bender" en la página 395.
- `BlendMode.SUBTRACT` (`"subtract"`): suele utilizarse para crear un efecto de disolución de oscurecimiento animado entre dos imágenes.

Ajuste de los colores de DisplayObject

Se pueden utilizar los métodos de la clase ColorTransform (flash.geom.ColorTransform) para ajustar el color de un objeto de visualización. Cada objeto de visualización tiene una propiedad `transform`, que es una instancia de la clase Transform y que contiene información sobre diversas transformaciones que se aplican al objeto de visualización (como la rotación, los cambios en la escala o posición, entre otros). Además de esta información sobre las transformaciones geométricas, la clase Transform también incluye una propiedad `colorTransform`, que es una instancia de la clase ColorTransform y que permite realizar ajustes de color en el objeto de visualización. Para acceder a la información sobre transformación de color de un objeto de visualización, se puede utilizar código como el siguiente:

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

Una vez creada una instancia de ColorTransform, se pueden leer sus valores de propiedades para saber cuáles son las transformaciones de color que ya se han aplicado o se pueden definir dichos valores para realizar cambios de color en el objeto de visualización. Para actualizar el objeto de visualización después de haber realizado cambios, es necesario reasignar la instancia de ColorTransform a la propiedad `transform.colorTransform`.

```
var colorInfo:ColorTransform = my DisplayObject.transform.colorTransform;

// Make some color transformations here.

// Commit the change.
myDisplayObject.transform.colorTransform = colorInfo;
```

Configuración de los valores de color a través del código

La propiedad `color` de la clase ColorTransform puede utilizarse para asignar un valor específico de color rojo, verde, azul (RGB) al objeto de visualización. En el ejemplo siguiente se usa la propiedad `color` para cambiar a azul el color del objeto de visualización denominado `square`, cuando el usuario haga clic en un botón denominado `blueBtn`:

```
// square is a display object on the Stage.
// blueBtn, redBtn, greenBtn, and blackBtn are buttons on the Stage.

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// Get access to the ColorTransform instance associated with square.
var colorInfo:ColorTransform = square.transform.colorTransform;

// This function is called when blueBtn is clicked.
function makeBlue(event:MouseEvent):void
{
    // Set the color of the ColorTransform object.
    colorInfo.color = 0x003399;
    // apply the change to the display object
    square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

Cuando se cambia el color de un objeto de visualización a través de la propiedad `color`, se cambia completamente el color de todo el objeto, independientemente de que el objeto tuviera varios colores. Por ejemplo, si hay un objeto de visualización que contiene un círculo verde con texto negro encima, al definir como una sombra de rojo la propiedad `color` de la instancia de ColorTransform asociada a dicho objeto, todo el objeto, círculo y texto, se volverá rojo (de modo que el texto ya no podrá distinguirse del resto del objeto).

Modificación de efectos de color y brillo a través del código

Supongamos que tiene un objeto de visualización con varios colores (por ejemplo, una foto digital) y no desea modificar los colores de todo el objeto, sino únicamente ajustar el color de un objeto de visualización basándose en los colores existentes. En esta situación, la clase ColorTransform incluye una serie de propiedades de multiplicador y desplazamiento que pueden utilizarse para realizar este tipo de ajuste. Las propiedades de multiplicador, denominadas `redMultiplier`, `greenMultiplier`, `blueMultiplier` y `alphaMultiplier`, funcionan como filtros fotográficos de color (o gafas de sol de color), ya que amplifican o reducen determinados colores en el objeto de visualización. Las propiedades de desplazamiento (`redOffset`, `greenOffset`, `blueOffset` y `alphaOffset`) pueden utilizarse para añadir cantidades adicionales de un determinado color al objeto o para especificar el valor mínimo que puede tener un determinado color.

Estas propiedades de multiplicador y desplazamiento son idénticas a la configuración avanzada de color disponible para los símbolos de clip de película en la herramienta de edición de Flash que se muestra al elegir Avanzado en el menú emergente Color del inspector de propiedades.

En el código siguiente se carga una imagen JPEG y se aplica una transformación de color que modifica los canales rojo y verde a medida que el puntero del ratón se mueve por los ejes x e y. En este caso, como no se especifica ningún valor de desplazamiento, el valor de cada uno de los canales de color mostrados en pantalla será un porcentaje del valor de color original de la imagen, lo que significa que la mayoría de rojo y verde mostrado en cualquier píxel será la cantidad original de rojo o verde de dicho píxel.

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// This function is called when the mouse moves over the loaded image.
function adjustColor(event:MouseEvent):void
{
    // Access the ColorTransform object for the Loader (containing the image)
    var colorTransformer:ColorTransform = loader.transform.colorTransform;

    // Set the red and green multipliers according to the mouse position.
    // The red value ranges from 0% (no red) when the cursor is at the left
    // to 100% red (normal image appearance) when the cursor is at the right.
    // The same applies to the green channel, except it's controlled by the
    // position of the mouse in the y axis.
    colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
    colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

    // Apply the changes to the display object.
    loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

Rotación de objetos

Los objetos de visualización pueden girarse a través de la propiedad `rotation`. Se puede leer este valor para saber si se ha girado un objeto. Para girar el objeto, se puede establecer esta propiedad en un número (en grados) que representa la cantidad de rotación que se aplica al objeto. Por ejemplo, en la siguiente línea de código se gira el objeto `square` 45 grados (un octavo de vuelta):

```
square.rotation = 45;
```

Para girar un objeto de visualización también se puede utilizar una matriz de transformación; esto se describe en "Trabajo con la geometría" en la página 351.

Aparición o desaparición progresiva de objetos

Se puede controlar la transparencia de un objeto de visualización para hacerlo transparente (parcialmente o en su totalidad) o cambiar la transparencia para hacer que el objeto aparezca o desaparezca progresivamente. La propiedad `alpha` de la clase DisplayObject define la transparencia (o, mejor dicho, la opacidad) de un objeto de visualización. La propiedad `alpha` puede definirse como cualquier valor entre 0 y 1, siendo 0 completamente transparente y 1 completamente opaco. Por ejemplo, con las siguientes líneas de código, el objeto denominado `myBall` se hace parcialmente transparente (en un 50 %) cuando se hace clic en él con el ratón:

```
function fadeBall(event:MouseEvent):void
{
    myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

También se puede modificar la transparencia de un objeto de visualización mediante los ajustes de color disponibles a través de la clase ColorTransform. Para obtener más información, consulte "Ajuste de los colores de DisplayObject" en la página 313.

Enmascarar objetos de visualización

Se puede utilizar un objeto de visualización como una máscara para crear un agujero a través del cual se ve el contenido de otro objeto de visualización.

Definición de una máscara

Para indicar que un objeto de visualización será la máscara de otro objeto de visualización, es preciso definir el objeto de máscara como la propiedad `mask` del objeto de visualización que se va a enmascarar:

```
// Make the object maskSprite be a mask for the object mySprite.
mySprite.mask = maskSprite;
```

El objeto de visualización con máscara aparece bajo todas las zonas opacas (no transparentes) del objeto de visualización que actúa como máscara. Por ejemplo, el código siguiente crea una instancia de Shape que contiene un cuadrado rojo de 100 por 100 píxeles y una instancia de Sprite que contiene un círculo azul con un radio de 25 píxeles. Cuando se hace clic en un círculo, se define como la máscara del cuadrado, de forma que la única parte del cuadrado que aparece visible es la que queda cubierta por la parte sólida del círculo. Dicho de otro modo, sólo se ve un círculo rojo.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.Shape;

// Draw a square and add it to the display list.
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// Draw a circle and add it to the display list.
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);

function maskSquare(event:MouseEvent):void
{
    square.mask = circle;
    circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

El objeto de visualización que actúa como máscara puede arrastrarse, animarse, cambiar de tamaño automáticamente y usar formas independientes en una sola máscara. No es necesario añadir el objeto de visualización de máscara a la lista de visualización. Sin embargo, si se desea ajustar la escala del objeto de máscara cuando se ajusta la escala del escenario, o permitir la interacción del usuario con la máscara (por ejemplo, el arrastre o cambio de tamaño controlado por el usuario), el objeto de máscara debe añadirse a la lista de visualización. El índice z real (de delante a atrás) de los objetos de visualización no es relevante, siempre y cuando el objeto de máscara se añada a la lista de visualización. (El objeto de máscara sólo aparecerá en la pantalla como una máscara.) Si el objeto de máscara es una instancia de MovieClip con varios fotogramas, reproduce todos ellos en su línea de tiempo, tal y como haría si no actuara como una máscara. Para eliminar una máscara, debe establecerse la propiedad `mask` en `null`:

```
// remove the mask from mySprite
mySprite.mask = null;
```

No puede utilizar una máscara para enmascarar otra máscara ni definir la propiedad `alpha` de un objeto de visualización de máscara. En un objeto de visualización que se usa como máscara, sólo se pueden utilizar los rellenos; los trazos se pasan por alto.

Enmascaramiento de fuentes de dispositivo

Se puede utilizar un objeto de visualización para enmascarar un texto definido en una fuente de dispositivo. Cuando se utiliza un objeto de visualización para enmascarar texto definido en una fuente de dispositivo, el recuadro de delimitación rectangular de la máscara se utiliza como la forma de máscara. Es decir, si se crea una máscara de objeto de visualización no rectangular para el texto de la fuente de dispositivo, la máscara que aparecerá en el archivo SWF tendrá la forma del recuadro de delimitación rectangular y no la de la máscara en sí.

Enmascaramiento del canal alfa

Se admite el enmascaramiento del canal alfa si tanto la máscara como los objetos de visualización enmascarados utilizan la caché de mapa de bits, como se muestra a continuación:

```
// maskShape is a Shape instance which includes a gradient fill.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

Una aplicación de enmascaramiento del canal alfa consiste, por ejemplo, en usar un filtro del objeto de máscara independientemente del filtro aplicado al objeto de visualización enmascarado.

En el siguiente ejemplo se carga un archivo de imagen externo en el escenario. Dicha imagen (mejor dicho, la instancia de Loader en la que se ubica) será el objeto de visualización que se enmascare. Se dibuja un óvalo con degradado (de color negro sólido en el centro y con transparencia progresiva en los bordes) sobre la imagen; será la máscara alfa. Ambos objetos de visualización tienen activada la caché de mapa de bits. El óvalo se define como una máscara de la imagen y luego se convierte en un elemento que se puede arrastrar.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// Load an image and add it to the display list.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create a Sprite.
var oval:Sprite = new Sprite();
// Draw a gradient oval.
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
var matrix:Matrix = new Matrix();
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                                colors,
                                alphas,
                                ratios,
                                matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// add the Sprite to the display list
this.addChild(oval);

// Set cacheAsBitmap = true for both display objects.
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// Set the oval as the mask for the loader (and its child, the loaded image)
loader.mask = oval;

// Make the oval draggable.
oval.startDrag(true);
```

Animación de objetos

La animación es el proceso de mover algo o, alternativamente, hacer que algo cambie con el tiempo. La animación mediante scripts es una parte fundamental de los videojuegos, que a menudo se utiliza para pulir y añadir útiles indicaciones de interacción a otras aplicaciones.

La idea fundamental en la que se basa la animación mediante scripts es la necesidad de realizar un cambio y de dividir dicho cambio en incrementos temporales. Repetir algo en ActionScript es tan sencillo como usar una sentencia de bucle común. Sin embargo, es necesario reproducir todas las repeticiones de un bucle para poder actualizar la visualización. Para crear animación mediante scripts, es necesario escribir código ActionScript que repita alguna acción a lo largo del tiempo y que además actualice la pantalla cada vez que se ejecuta.

Por ejemplo, imagine que desea crear una sencilla animación de una bola que se desplaza a través de la pantalla. ActionScript incluye un mecanismo sencillo que permite seguir el paso del tiempo y actualizar la pantalla de forma correspondiente, lo que significa que se puede escribir código que mueva la bola un poco cada vez hasta llegar a su destino. La pantalla se actualiza después de cada movimiento, de forma que el movimiento a través del escenario es visible para el espectador.

Desde un punto de vista práctico, tiene sentido sincronizar la animación mediante scripts con la velocidad de fotogramas del archivo SWF (es decir, hacer que una animación cambie cada vez que se muestra un nuevo fotograma), ya que define la frecuencia con la que Flash Player o AIR actualiza la pantalla. Cada objeto de visualización tiene un evento `enterFrame` que se distribuye según la velocidad de fotogramas del archivo SWF: un evento por fotograma. La mayoría de los desarrolladores que crean animación mediante scripts utilizan el evento `enterFrame` para crear acciones que se repiten a lo largo del tiempo. Se puede escribir un código que detecte el evento `enterFrame` y mueva la bola animada una cantidad determinada en cada fotograma y, al actualizarse la pantalla (en cada fotograma), redibuje la bola en la nueva ubicación, creando movimiento.

Nota: *otra forma de repetir una acción a lo largo del tiempo consiste en utilizar la clase Timer. Una instancia de Timer activa una notificación de evento cada vez que pasa un tiempo determinado. Se puede escribir código que lleve a cabo la animación mediante la gestión del evento timer de la clase Timer, si se define un intervalo de tiempo pequeño (alguna fracción de un segundo). Para obtener más información sobre el uso de la clase Timer, consulte "Control de intervalos de tiempo" en la página 138.*

En el ejemplo siguiente, se crea en el escenario una instancia de Sprite circular, denominada `circle`. Cuando el usuario hace clic en el círculo, se inicia una secuencia de animación mediante scripts que provoca la desaparición progresiva de `circle` (se reduce su propiedad `alpha`) hasta que es completamente transparente:

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// draw a circle and add it to the display list
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// When this animation starts, this function is called every frame.
// The change made by this function (updated to the screen every
// frame) is what causes the animation to occur.
function fadeCircle(event:Event):void
{
    circle.alpha -= .05;

    if (circle.alpha <= 0)
    {
        circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
    }
}

function startAnimation(event:MouseEvent):void
{
    circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

Cuando el usuario hace clic en el círculo, se suscribe la función `fadeCircle()` como detector del evento `enterFrame`, lo que significa que se inicia su llamada una vez en cada fotograma. Esa función hace que `circle` desaparezca progresivamente cambiando su propiedad `alpha`, de modo que, una vez en cada fotograma, el valor de `alpha` del círculo se reduce en 0,05 (5 por ciento) y se actualiza la pantalla. Finalmente, cuando el valor de `alpha` es 0 (`circle` es completamente transparente), se elimina la función `fadeCircle()` como detector de eventos y finaliza la animación.

Este mismo código podría utilizarse, por ejemplo, para crear un movimiento de animación en lugar de la desaparición progresiva. Si se sustituye otra propiedad de `alpha` en la función que actúa como detector del evento `enterFrame`, esa propiedad se animará. Por ejemplo, si se cambia esta línea

```
circle.alpha -= .05;
```

por este código

```
circle.x += 5;
```

se animará la propiedad `x`, haciendo que el círculo se mueva hacia la derecha a través del escenario. La condición que finaliza la animación puede cambiarse para que la animación termine (es decir, se quite la suscripción del detector de `enterFrame`) al llegar a la coordenada x especificada.

Carga dinámica de contenido de visualización

En una aplicación de ActionScript 3.0, se puede cargar cualquiera de los siguientes activos de visualización externos:

- Un archivo SWF creado en ActionScript 3.0: este archivo puede ser un objeto Sprite, MovieClip o de cualquier clase que amplíe Sprite.
- Un archivo de imagen: por ejemplo, archivos JPG, PNG y GIF.
- Un archivo SWF AVM1: un archivo SWF escrito en ActionScript 1.0 ó 2.0.

Estos activos se cargan mediante la clase Loader.

Carga de objetos de visualización

Los objetos Loader se usan para cargar archivos SWF y archivos de gráficos en una aplicación. La clase Loader es una subclase de la clase DisplayObjectContainer. Un objeto Loader sólo puede contener un objeto de visualización secundario en su lista de visualización: el objeto de visualización que representa el archivo SWF o archivo de gráficos que se carga. Cuando se añade un objeto Loader a la lista de visualización, como en el código siguiente, también se añade a la lista de visualización el objeto de visualización secundario cargado, una vez que se ha cargado:

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

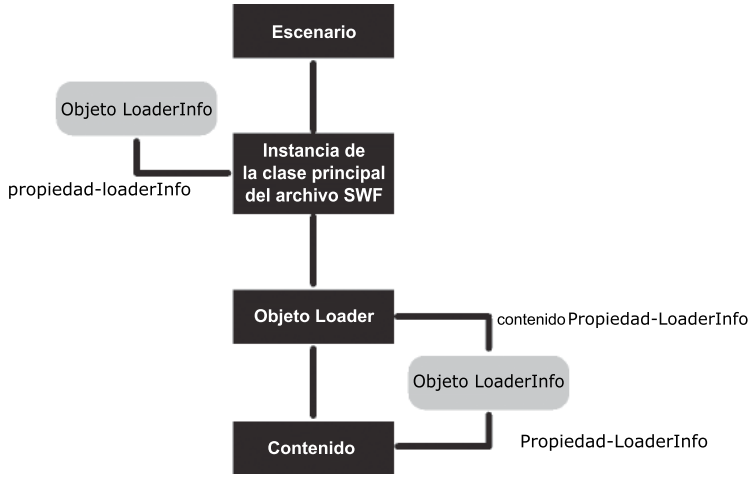
Cuando se carga el archivo SWF o la imagen, se puede mover el objeto de visualización cargado a otro contenedor de objeto de visualización, como el objeto DisplayObjectContainer `container` que se muestra en este ejemplo:

```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
    container.addChild(pictLdr.content);
}
```

Supervisión del progreso de carga

Cuando se empieza a cargar el archivo, se crea un objeto LoaderInfo. Un objeto LoaderInfo proporciona información tal como el progreso de carga, los URL del cargador y el contenido cargado, el número total de bytes del medio, y la anchura y la altura nominal del medio. Un objeto LoaderInfo también distribuye eventos para supervisar el progreso de la carga.

El siguiente diagrama muestra los diferentes usos del objeto LoaderInfo para la instancia de la clase principal del archivo SWF, para un objeto Loader y para un objeto cargado por el objeto Loader:



Se puede acceder al objeto LoaderInfo como una propiedad tanto del objeto Loader como del objeto de visualización cargado. En cuanto comienza la carga, se puede acceder al objeto LoaderInfo a través de la propiedad `contentLoaderInfo` del objeto Loader. Cuando finaliza la carga del objeto de visualización, también es posible acceder al objeto LoaderInfo como una propiedad del objeto de visualización cargado, a través de la propiedad `loaderInfo` del objeto de visualización. La propiedad `loaderInfo` del objeto de visualización hace referencia al mismo objeto LoaderInfo al que se refiere la propiedad `contentLoaderInfo` del objeto Loader. Dicho de otro modo, un objeto LoaderInfo se comparte entre un objeto cargado y el objeto Loader que lo ha cargado (es decir, entre el contenido cargado y el cargador).

Para acceder a las propiedades del contenido cargado, es necesario añadir un detector de eventos al objeto LoaderInfo, como se muestra en el siguiente código:

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
    var content:Sprite = event.target.content;
    content.scaleX = 2;
}
```

Para obtener más información, consulte "Gestión de eventos" en la página 254.

Especificación del contexto de carga

Cuando se carga un archivo externo en Flash Player o AIR con el método `load()` o `loadBytes()`, se puede especificar opcionalmente un parámetro `context`. Este parámetro es un objeto LoaderContext.

La clase LoaderContext incluye tres propiedades que permiten definir el contexto de uso del contenido cargado:

- `checkPolicyFile`: utilice esta propiedad sólo si carga un archivo de imagen (no un archivo SWF). Si establece esta propiedad en `true`, Loader comprueba el servidor de origen de un archivo de política (consulte "Controles de sitio Web (archivos de política)" en la página 721). Sólo es necesaria en el contenido procedente de dominios ajenos al del archivo SWF que contiene el objeto Loader. Si el servidor concede permisos al dominio de Loader, el código ActionScript de los archivos SWF del dominio de Loader puede acceder a los datos de la imagen cargada y, por lo tanto, se puede usar el comando `BitmapData.draw()` para acceder a los datos de la imagen cargada.

  Tenga en cuenta que un archivo SWF de otros dominios ajenos al del objeto Loader puede llamar a `Security.allowDomain()` para permitir el uso de un dominio específico.

- `securityDomain`: utilice esta propiedad sólo si carga un archivo SWF (no una imagen). Esta propiedad se especifica en un archivo SWF de un dominio ajeno al del archivo que contiene el objeto Loader. Si se especifica esta opción, Flash Player comprueba la existencia de un archivo de política y, en caso de que exista uno, los archivos SWF de los dominios permitidos en el archivo de política entre dominios pueden reutilizar los scripts del contenido SWF cargado. Se puede especificar `flash.system.SecurityDomain.currentDomain` como este parámetro.

- `applicationDomain`: utilice esta propiedad solamente si carga un archivo SWF escrito en ActionScript 3.0 (no una imagen ni un archivo SWF escritos en ActionScript 1.0 ó 2.0). Al cargar el archivo, se puede especificar que se incluya en el mismo dominio de aplicación del objeto Loader; para ello, hay que establecer el parámetro `applicationDomain` en `flash.system.ApplicationDomain.currentDomain`. Al colocar el archivo SWF cargado en el mismo dominio de aplicación, se puede acceder a sus clases directamente. Esto puede ser muy útil si se carga un archivo SWF que contiene medios incorporados, a los que se puede tener acceso a través de sus nombres de clase asociados. Para obtener más información, consulte "Utilización de la clase ApplicationDomain" en la página 667.

A continuación se muestra un ejemplo de comprobación de un archivo de política durante la carga de un mapa de bits de otro dominio:

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

A continuación se muestra un ejemplo de comprobación de un archivo de política durante la carga de un archivo SWF de otro dominio, con el fin de colocar el archivo en el mismo entorno limitado de seguridad que el objeto Loader. Además, el código añade las clases del archivo SWF cargado al mismo dominio de aplicación que el del objeto Loader:

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

Para obtener más información, consulte la clase LoaderContext en la Referencia del lenguaje y componentes ActionScript 3.0.

Ejemplo: SpriteArranger

La aplicación de ejemplo SpriteArranger se basa en la aplicación de ejemplo de formas geométricas que se describe por separado (consulte “Ejemplo: GeometricShapes” en la página 126).

La aplicación de ejemplo SpriteArranger ilustra diversos conceptos relacionados con los objetos de visualización:

- Ampliación de las clases de objetos de visualización
- Añadir objetos a la lista de visualización
- Organización en capas de los objetos de visualización y utilización de contenedores de objetos de visualización
- Respuesta a eventos de objetos de visualización
- Utilización de propiedades y métodos de objetos de visualización

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación SpriteArranger se encuentran en la carpeta Examples/SpriteArranger. La aplicación consta de los siguientes archivos:

Archivo	Descripción
SpriteArranger.mxml o SpriteArranger.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/SpriteArranger/CircleSprite.as	Una clase que define un tipo de objeto Sprite que representa un círculo en pantalla.
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	Una clase que define el lienzo, que es un contenedor de objeto de visualización que contiene objetos GeometricSprite.
com/example/programmingas3/SpriteArranger/SquareSprite.as	Una clase que define un tipo de objeto Sprite que representa un cuadrado en pantalla.
com/example/programmingas3/SpriteArranger/TriangleSprite.as	Una clase que define un tipo de objeto Sprite que representa un triángulo en pantalla.

Archivo	Descripción
com/example/programmingas3/SpriteArranger/GeometricSprite.as	Una clase que amplía el objeto Sprite y que se usa para definir una forma en pantalla. CircleSprite, SquareSprite y TriangleSprite amplían esta clase.
com/example/programmingas3/geometricshapes/IGeometricShape.as	La interfaz base que define los métodos que implementarán todas las clases de formas geométricas.
com/example/programmingas3/geometricshapes/IPolygon.as	Una interfaz que define los métodos que implementarán las clases de formas geométricas que tienen varios lados.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Un tipo de forma geométrica que tiene lados de igual longitud, simétricamente ubicados alrededor del centro de la forma.
com/example/programmingas3/geometricshapes/Circle.as	Un tipo de forma geométrica que define un círculo.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Una subclase de RegularPolygon que define un triángulo con todos los lados de la misma longitud.
com/example/programmingas3/geometricshapes/Square.as	Una subclase de RegularPolygon que define un rectángulo con los cuatro lados de la misma longitud.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Una clase que contiene un "método de fábrica" para crear formas a partir de un tipo y tamaño de forma.

Definición de las clases de SpriteArranger

La aplicación SpriteArranger permite al usuario añadir varios objetos de visualización al "lienzo" en pantalla.

La clase DrawingCanvas define un área de dibujo, un tipo de contenedor de objeto de visualización, en el que el usuario puede añadir formas en pantalla. Estas formas en pantalla son instancias de una de las subclases de la clase GeometricSprite.

La clase DrawingCanvas

La clase DrawingCanvas amplía la clase Sprite y su herencia se define en la declaración de la clase DrawingCanvas de la manera siguiente:

```
public class DrawingCanvas extends Sprite
```

La clase Sprite es una subclase de las clases DisplayObjectContainer y DisplayObject, y la clase DrawingCanvas utiliza los métodos y las propiedades de dichas clases.

El método constructor `DrawingCanvas()` configura un objeto Rectangle, `bounds`, que se utiliza para dibujar el contorno del lienzo. A continuación, llama al método `initCanvas()`, del siguiente modo:

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

Tal y como se muestra en el ejemplo siguiente, el método `initCanvas()` define diversas propiedades del objeto DrawingCanvas que se pasaron como argumentos a la función constructora:

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

El método `initCanvas()` llama entonces al método `drawBounds()`, que dibuja el lienzo a través de la propiedad `graphics` de DrawingCanvas. La propiedad `graphics` se hereda de la clase Shape.

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
                       bounds.top - 1,
                       bounds.width + 2,
                       bounds.height + 2);
this.graphics.endFill();
```

Los siguientes métodos adicionales de la clase DrawingCanvas se llaman en función de las interacciones del usuario con la aplicación:

- Los métodos `addShape()` y `describeChildren()`, que se describen en "Añadir objetos de visualización al lienzo" en la página 326
- Los métodos `moveToBack()`, `moveDown()`, `moveToFront()` y `moveUp()`, descritos en "Reorganización de las capas de los objetos de visualización" en la página 328
- El método `onMouseUp()`, descrito en "Selección mediante clic y arrastre de objetos de visualización" en la página 327

La clase GeometricSprite y sus subclases

Cada objeto de visualización que puede añadir el usuario al lienzo es una instancia de una de las siguientes subclases de la clase GeometricSprite:

- CircleSprite
- SquareSprite
- TriangleSprite

La clase GeometricSprite amplía la clase flash.display.Sprite:

```
public class GeometricSprite extends Sprite
```

La clase GeometricSprite incluye varias propiedades comunes para todos los objetos GeometricSprite. Estas propiedades se definen en la función constructora según los parámetros pasados a la función. Por ejemplo:

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

La propiedad `geometricShape` de la clase GeometricSprite define una interfaz IGeometricShape, que define las propiedades matemáticas de la forma, pero no las propiedades visuales. Las clases que implementan la interfaz IGeometricShape se definen en la aplicación de ejemplo GeometricShapes (consulte "Ejemplo: GeometricShapes" en la página 126).

La clase GeometricSprite define el método `drawShape()`, que se refina en las definiciones de sustitución de cada una de las subclases de GeometricSprite. Para obtener más información, consulte la siguiente sección "Añadir objetos de visualización al lienzo".

La clase GeometricSprite también proporciona los siguientes métodos:

- Los métodos `onMouseDown()` y `onMouseUp()`, que se describen en "Selección mediante clic y arrastre de objetos de visualización" en la página 327

- Los métodos `showSelected()` y `hideSelected()`, descritos en "Selección mediante clic y arrastre de objetos de visualización" en la página 327

Añadir objetos de visualización al lienzo

Cuando el usuario hace clic en el botón Añadir forma, la aplicación llama al método `addShape()` de la clase DrawingCanvas. Crea una instancia de una nueva clase GeometricSprite mediante una llamada a la función constructora adecuada de una de las subclases de GeometricSprite, como se muestra en el siguiente ejemplo:

```
public function addShape(shapeName:String, len:Number):void
{
    var newShape:GeometricSprite;
    switch (shapeName)
    {
        case "Triangle":
            newShape = new TriangleSprite(len);
            break;

        case "Square":
            newShape = new SquareSprite(len);
            break;

        case "Circle":
            newShape = new CircleSprite(len);
            break;
    }
    newShape.alpha = 0.8;
    this.addChild(newShape);
}
```

Cada método constructor llama al método `drawShape()`, que utiliza la propiedad `graphics` de la clase (heredada de la clase Sprite) para dibujar el gráfico vectorial adecuado. Por ejemplo, el método `drawShape()` de la clase CircleSprite incluye el siguiente código:

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

La penúltima línea de la función `addShape()` define la propiedad `alpha` del objeto de visualización (heredada de la clase DisplayObject), de forma que cada objeto de visualización que se añade al lienzo es ligeramente transparente, lo que permite que el usuario vea los objetos que hay detrás.

La línea final del método `addChild()` añade el nuevo objeto de visualización a la lista de elementos secundarios de la instancia de la clase DrawingCanvas, que ya se encuentra en la lista de visualización. Como resultado, el nuevo objeto de visualización aparece en el escenario.

La interfaz de la aplicación incluye dos campos de texto: `selectedSpriteTxt` y `outputTxt`. Las propiedades de texto de estos campos se actualizan con la información relativa a los objetos GeometricSprite que se han añadido al lienzo o que ha seleccionado el usuario. La clase GeometricSprite gestiona esta tarea de notificación de información mediante la sustitución del método `toString()`, del siguiente modo:

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " + this.y;
}
```

La propiedad `shapeType` se establece en el valor adecuado en el método constructor de cada subclase de GeometricSprite. Por ejemplo, el método `toString()` podría devolver el siguiente valor de una instancia de CircleSprite recién añadida a la instancia de DrawingCanvas:

```
Circle of size 50 at 0, 0
```

El método `describeChildren()` de la clase DrawingCanvas reproduce indefinidamente la lista de elementos secundarios del lienzo, utilizando la propiedad `numChildren` (heredada de la clase DisplayObjectContainer) para definir el límite del bucle `for`. Genera una cadena con el valor de cada elemento secundario, del siguiente modo:

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
child = this.getChildAt(i);
desc += i + ": " + child + '\n';
}
```

La cadena resultante se utiliza para establecer la propiedad `text` del campo de texto `outputTxt`.

Selección mediante clic y arrastre de objetos de visualización

Cuando el usuario hace clic en una instancia de GeometricSprite, la aplicación llama al controlador de eventos `onMouseDown()`. Como se muestra a continuación, se define este controlador de eventos para detectar los eventos de clic del ratón en la función constructora de la clase GeometricSprite:

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

A continuación, el método `onMouseDown()` llama al método `showSelected()` del objeto GeometricSprite. Si es la primera vez que se ha llamado a este método en el objeto, el método crea un nuevo objeto Shape denominado `selectionIndicator` y utiliza la propiedad `graphics` del objeto Shape para dibujar un rectángulo de resaltado rojo, del siguiente modo:

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1, this.size + 1);
this.addChild(this.selectionIndicator);
```

Si no es la primera vez que se llama al método `onMouseDown()`, el método simplemente establece la propiedad `visible` del objeto Shape `selectionIndicator`(heredada de la clase DisplayObject), del siguiente modo:

```
this.selectionIndicator.visible = true;
```

El método `hideSelected()` oculta el objeto Shape `selectionIndicator` del objeto seleccionado previamente estableciendo su propiedad `visible` en `false`.

El método del controlador de eventos `onMouseDown()` también llama al método `startDrag()` (heredado de la clase Sprite), que incluye el siguiente código:

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

De esta forma, el usuario puede arrastrar el objeto seleccionado por el lienzo, dentro de los límites definidos por el rectángulo `boundsRect`.

Cuando el usuario suelta el botón del ratón, se distribuye el evento `mouseUp`. El método constructor de DrawingCanvas configura el siguiente detector de eventos:

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

Este detector de eventos se define en el objeto DrawingCanvas y no en cada uno de los objetos GeometricSprite. El motivo de ello es que, al arrastrar el objeto GeometricSprite, éste podría acabar detrás de otro objeto de visualización (otro objeto GeometricSprite) al soltar el botón del ratón. El objeto de visualización en primer plano recibiría el evento de soltar el ratón pero no así el objeto de visualización que arrastra el usuario. Al añadir el detector al objeto DrawingCanvas, se garantiza un control perpetuo del evento.

El método `onMouseUp()` llama al método `onMouseUp()` del objeto GeometricSprite, que a su vez llama al método `stopDrag()` del objeto GeometricSprite.

Reorganización de las capas de los objetos de visualización

La interfaz de usuario de la aplicación incluye botones Move Back, Move Down, Move Up y Move to Front para mover atrás, abajo, arriba y al frente, respectivamente. Cuando el usuario hace clic en uno de estos botones, la aplicación llama al método correspondiente de la clase DrawingCanvas: `moveToBack()`, `moveDown()`, `moveUp()` o `moveToFront()`. Por ejemplo, el método `moveToBack()` incluye el siguiente código:

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

El método utiliza el método `setChildIndex()` (heredado de la clase DisplayObjectContainer) para colocar el objeto de visualización en la posición de índice 0 en la lista de elementos secundarios de la instancia de DrawingCanvas (`this`).

El método `moveDown()` funciona de forma similar, excepto en que reduce en 1 la posición de índice del objeto de visualización en la lista de elementos secundarios de la instancia de DrawingCanvas:

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

Los métodos `moveUp()` y `moveToFront()` funcionan de modo similar a los métodos `moveToBack()` y `moveDown()`.

Capítulo 14: Utilización de la API de dibujo

Si bien las imágenes e ilustraciones importadas tienen una gran relevancia, la funcionalidad conocida como la API de dibujo, que permite dibujar líneas y formas en ActionScript, ofrece la libertad de iniciar una aplicación con el equipo en lo que sería el equivalente a un lienzo en blanco en el que se pueden crear las imágenes que se deseen. La capacidad de crear gráficos propios abre un amplísimo abanico de posibilidades para las aplicaciones. Gracias a las técnicas que se tratarán en este capítulo, es posible crear un programa de dibujo, diseñar obras de arte animadas e interactivas o desarrollar mediante programación elementos de interfaz propios, entre muchas otras posibilidades.

Fundamentos de la utilización de la API de dibujo

Introducción a la utilización de la API de dibujo

La funcionalidad integrada en ActionScript que permite crear gráficos vectoriales (líneas, curvas, formas, rellenos y degradados) y mostrarlos en la pantalla mediante ActionScript recibe el nombre de API de dibujo. La clase flash.display.Graphics es la encargada de proporcionar esta funcionalidad. Con ActionScript es posible dibujar en cualquier instancia de Shape, Sprite o MovieClip utilizando la propiedad `graphics` definida en cada una de esas clases. (La propiedad `graphics` de cada una de esas clases es en realidad una instancia de la clase Graphics.)

A la hora de empezar a dibujar mediante código, la clase Graphics ofrece varios métodos que facilitan el trazado de formas comunes, como círculos, elipses, rectángulos y rectángulos con esquinas redondeadas. Todas ellas se pueden dibujar como líneas vacías o como formas rellenas. Cuando sea necesaria una funcionalidad más avanzada, la clase Graphics también dispone de métodos para trazar líneas y curvas cuadráticas de Bézier, que se pueden emplear junto con las funciones trigonométricas de la clase Math para crear cualquier forma imaginable.

Flash Player 10 y Adobe AIR 1.5 incorporan una API adicional para dibujo que permite dibujar mediante programación formas completas con un único comando. Un vez que se haya familiarizado con la clase Graphics y las tareas incluidas en “Fundamentos de la utilización de la API de dibujo”, continúe con “Utilización avanzada de la API de dibujo” en la página 342 para obtener más información sobre estas funciones de la API de dibujo.

Tareas comunes de la API de dibujo

A continuación se presenta un conjunto de tareas que suelen realizarse mediante la API de dibujo de ActionScript y que se estudiarán en este capítulo:

- Definir estilos de línea y de relleno para dibujar formas
- Dibujar líneas rectas y curvas
- Usar métodos para dibujar formas como círculos, elipses y rectángulos
- Dibujar con líneas de degradados y rellenos
- Definir una matriz para crear un degradado
- Usar funciones trigonométricas con la API de dibujo

- Incorporar la API de dibujo en una animación

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Punto de anclaje: uno de los dos puntos finales de una curva cuadrática de Bézier.
- Punto de control: punto que define la dirección y la curvatura de una curva cuadrática de Bézier. La línea curva nunca alcanza el punto de control, no obstante, se curva como si fuese arrastrada hacia éste.
- Espacio de coordenadas; gráfica de coordenadas contenida en un objeto de visualización sobre la que se colocan sus elementos secundarios.
- Relleno: parte interior sólida de una forma que tiene una línea rellena con color o la totalidad de una forma que no tiene contorno.
- Degradado: color que está formado por la transición gradual de un color a otro u otros (a diferencia de un color sólido).
- Punto: ubicación individual en un espacio de coordenadas. En el sistema de coordenadas bidimensional utilizado en ActionScript, el punto se define por su ubicación en el eje x y en el eje y (las coordenadas del punto).
- Curva cuadrática de Bézier: tipo de curva definida por una fórmula matemática concreta. En este tipo de curva, la forma se calcula basándose en las posiciones de los puntos de ancla (los extremos de la curva) y el punto de control que define la dirección y la curvatura del trazo.
- Escala: tamaño de un objeto en relación con su tamaño original. Ajustar la escala de un objeto significa cambiar su tamaño estirándolo o encogiéndolo.
- Trazo: parte del contorno de una forma que tiene una línea rellena con un color o las líneas de una forma sin relleno.
- Trasladar: cambiar las coordenadas de un punto de un espacio de coordenadas a otro.
- Eje X: eje horizontal del sistema de coordenadas bidimensional que se utiliza en ActionScript.
- Eje Y: eje vertical del sistema de coordenadas bidimensional que se utiliza en ActionScript.

Ejecución de los ejemplos del capítulo

A medida que progrese en el estudio de este capítulo, podría desear probar algunos de los listados de código de ejemplo. Debido a que este capítulo se centra en el dibujo de contenido visual, la prueba de los listados de código implica su ejecución y ver los resultados en el archivo SWF creado. Para probar los listados de código:

1. Cree un documento de Flash vacío.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Ejecute el programa utilizando Control > Probar película.

   Podrá ver el resultado del listado de código en el archivo SWF creado.

Aspectos básicos de la clase Graphics

Todos los objetos Shape, Sprite y MovieClip tienen una propiedad `graphics`, que es una instancia de la clase Graphics. La clase Graphics incluye propiedades y métodos para dibujar líneas, rellenos y formas. Cuando se necesita un objeto de visualización que se va a utilizar únicamente como un lienzo para dibujar contenido, se puede utilizar una instancia de Shape. Las instancias de Shape funcionan mejor que otros objetos de visualización a la hora de dibujar porque no tienen la sobrecarga de la funcionalidad adicional con la que cuentan las clases Sprite y MovieClip. En caso de que se desee un objeto de visualización en el que se pueda dibujar contenido gráfico y que, al mismo tiempo, pueda contener otros objetos de visualización, se puede utilizar una instancia de Sprite. Para más información sobre la forma de determinar qué objeto de visualización debe usarse para las distintas tareas, consulte "Selección de una subclase DisplayObject" en la página 299.

Dibujo de líneas y curvas

Todos los dibujos que se pueden realizar con una instancia de Graphics se basan en trazados básicos con líneas y curvas. En consecuencia, todos los dibujos con ActionScript deben llevarse a cabo utilizando la misma serie de pasos:

- Definir los estilos de línea y relleno.
- Establecer la posición inicial de dibujo.
- Dibujar líneas, curvas y formas (moviendo opcionalmente el punto de dibujo).
- En caso de ser necesario, finalizar creando un relleno.

Definición de los estilos de línea y relleno

Para dibujar con la propiedad `graphics` de una instancia de Shape, Sprite o MovieClip, en primer lugar es necesario definir el estilo (tamaño y color de línea, color de relleno) que se utilizará al dibujar. Al igual que cuando se utilizan las herramientas de dibujo de Adobe® Flash® CS4 Professional u otra aplicación de dibujo, al utilizar ActionScript se puede dibujar con o sin trazo y con o sin color de relleno. La apariencia del trazo se especifica mediante los métodos `lineStyle()` o `lineGradientStyle()`. Para crear una línea sólida se emplea el método `lineStyle()`. Al llamar a este método, los valores más comunes que se especificarán son los primeros tres parámetros: grosor de línea, color y alfa. Por ejemplo, la siguiente línea de código indica al objeto Shape llamado `myShape` que dibuje líneas de 2 píxeles de grosor, de color rojo (0x990000) y de una opacidad del 75%:

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

El valor predeterminado del parámetro alfa es de 1,0 (100%), de modo que se puede omitir el parámetro si se desea una línea completamente opaca. El método `lineStyle()` también acepta dos parámetros más para los consejos de píxeles y el modo de escala. Para más información sobre el uso de estos parámetros, consulte la descripción del método `Graphics.lineStyle()` en la Referencia del lenguaje y componentes ActionScript 3.0.

Para crear una línea con gradiente se emplea el método `lineGradientStyle()`. Este método se describe en "Creación de líneas y rellenos degradados" en la página 334.

Para crear una forma con relleno es necesario llamar a los métodos `beginFill()`, `beginGradientFill()`, `beginBitmapFill()` o `beginShaderFill()` antes de empezar a dibujar. El más básico de ellos, el método `beginFill()`, acepta dos parámetros: el color de relleno y (opcionalmente) un valor de alfa para el color de relleno. Por ejemplo, para dibujar una forma con un relleno verde sólido, se puede usar el siguiente código (suponiendo que se vaya a dibujar un objeto llamado `myShape`):

```
myShape.graphics.beginFill(0x00FF00);
```

Llamar a cualquier método de relleno termina de forma implícita cualquier relleno antes de iniciar uno nuevo. Llamar a cualquier método que especifique un estilo de trazo sustituye al trazo anterior, pero no modifica el relleno especificado previamente (y viceversa).

Una vez indicado el estilo de línea y las propiedades de relleno, el siguiente paso es especificar el punto de inicio del dibujo. La instancia de Graphics tiene un punto de dibujo, como la punta de un lápiz sobre la superficie de un papel. La siguiente acción de dibujo empezará allá donde se encuentre el punto de dibujo. Inicialmente, los objetos Graphics tienen su punto de dibujo en el punto 0,0 del espacio de coordenadas del objeto en el que se está dibujando. Para empezar a dibujar en un punto diferente es necesario llamar primero al método `moveTo()` antes de llamar a cualquiera de los métodos de dibujo. Esto es análogo a levantar la punta del lápiz del papel y moverla a una nueva posición.

Una vez que el punto de dibujo está colocado en su lugar, se empieza a dibujar utilizando una serie de llamadas a los métodos de dibujo `lineTo()` (para dibujar líneas rectas) y `curveTo()` (para dibujar curvas).

Mientras se dibuja es posible llamar al método `moveTo()` en cualquier momento para mover el punto de dibujo a una nueva posición sin dibujar.

Durante el dibujo, si se ha especificado un color de relleno, se puede indicar a Adobe Flash Player o Adobe® AIR™ que cierre el relleno llamando al método `endFill`. Si no se ha dibujado una forma cerrada (dicho de otro modo, si en el momento en que se llama a `endFill()` el punto de dibujo no se encuentra en el punto de inicio de la forma), al llamar al método `endFill()`, Flash Player o AIR cerrará automáticamente la forma dibujando una línea recta desde el punto de dibujo actual hasta la ubicación especificada en la llamada a `moveTo()` más reciente. Si se ha iniciado un relleno y no se ha llamado a `endFill()`, al llamar a `beginFill()` (o a cualquier otro método de relleno) se cerrará el relleno actual y se iniciará uno nuevo.

Dibujo de líneas rectas

Al llamar al método `lineTo()`, el objeto Graphics dibuja una línea recta desde el punto de dibujo actual hasta las coordenadas especificadas en los dos parámetros de la llamada al método usando el estilo de línea que se haya indicado. Por ejemplo, la siguiente línea de código coloca el punto de dibujo en las coordenadas 100, 100 y, a continuación, dibuja una línea hasta el punto 200, 200:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

El siguiente ejemplo dibuja triángulos de color rojo y verde con una altura de 100 píxeles:

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight / 2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight / 2, 0);

// green triangle, starting at point 200, 0
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight / 2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight / 2, 0);

this.addChild(triangle);
```

Dibujo de curvas

El método `curveTo()` dibuja una curva cuadrática de Bézier, es decir, un arco que conecta dos puntos (llamados puntos de ancla) y se inclina hacia un tercero (llamado punto de control). El objeto Graphics utiliza la posición de dibujo actual como primer punto de ancla. Al llamar al método `curveTo()`, se pasan cuatro parámetros: las coordenadas x e y, seguidas por las coordenadas x e y del segundo punto de ancla. Por ejemplo, el siguiente código dibuja una curva que empieza en el punto 100, 100 y termina en el punto 200, 200. Dado que el punto de control se encuentra en 175, 125, la curva que se genera se mueve hacia la derecha y luego hacia abajo:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

El siguiente ejemplo dibuja objetos circulares rojos y verdes con una altura y una anchura de 100 píxeles. Hay que tener en cuenta que, debido a la naturaleza de la ecuación cuadrática de Bézier, no son círculos perfectos:

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);

// green circular shape
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

Dibujo de formas mediante los métodos incorporados

Por comodidad, a la hora de dibujar formas comunes, como círculos, elipses, rectángulos y rectángulos con esquinas redondeadas, ActionScript 3.0 dispone de métodos para dibujar esas formas automáticamente. Más concretamente, son los métodos `drawCircle()`, `drawEllipse()`, `drawRect()`, `drawRoundRect()` y `drawRoundRectComplex()` de la clase Graphics los encargados de realizar dichas tareas. Estos métodos se pueden usar en lugar de `lineTo()` y `curveTo()`. Sin embargo, hay que tener en cuenta que sigue siendo necesario especificar los estilos de línea y relleno antes de llamar a estos métodos.

A continuación, se vuelve a usar el ejemplo del dibujo de cuadrados rojos, verdes y azules con una altura y una anchura de 100 píxeles. Este código utiliza el método `drawRect()` y, además, especifica que el color de relleno tiene un valor alfa del 50% (0.5):

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

En un objeto Sprite o MovieClip, el contenido de dibujo creado con la propiedad `graphics` siempre aparece detrás de todos los objetos de visualización secundarios contenidos en el objeto. Además, el contenido de la propiedad `graphics` no es un objeto de visualización independiente, de modo que no aparece en la lista de un elemento secundario de un objeto Sprite o MovieClip. Por ejemplo, el siguiente objeto Sprite tiene un círculo dibujado con su propiedad `graphics` y un objeto TextField en su lista de objetos de visualización secundarios:

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

Hay que tener en cuenta que el objeto TextField aparece sobre el círculo dibujado con el objeto graphics.

Creación de líneas y rellenos degradados

El objeto graphics también puede dibujar trazos y rellenos con degradados en lugar de con colores sólidos. Los trazos con degradado se crean mediante el método `lineGradientStyle()`, mientras que para los rellenos degradados se usa el método `beginGradientFill()`.

Ambos métodos aceptan los mismos parámetros. Los cuatro primeros son necesarios: tipo, colores, alfas y proporciones. Los cuatro restantes son opcionales, pero resultan útiles para llevar a cabo una personalización avanzada.

- El primer parámetro especifica el tipo de degradado que se va a crear. Los valores válidos son `GradientFill.LINEAR` o `GradientFill.RADIAL`.
- El segundo parámetro especifica el conjunto de valores de color que se usará. En un degradado lineal, los colores se ordenarán de izquierda a derecha. En un degradado radial, se organizarán de dentro a fuera. El orden de los colores del conjunto representa el orden en el que los colores se dibujarán en el degradado.
- El tercer parámetro especifica los valores de transparencia alfa de los colores correspondientes al parámetro anterior.
- El cuarto parámetro especifica las proporciones, o el énfasis que cada color tiene dentro del gradiente. Los valores válidos se encuentran entre 0 y 255. Estos valores no representan una altura o anchura, sino una posición dentro del degradado; 0 representa el inicio del degradado y 255 el final. El conjunto de proporciones debe aumentar secuencialmente y tener el mismo número de entradas que los conjuntos de color y de alfa especificados en el segundo y tercer parámetros.

Si bien el quinto parámetro, la matriz de transformación, es opcional, se suele utilizar con frecuencia, ya que constituye una forma fácil y potente de controlar la apariencia del degradado. Este parámetro acepta una instancia de Matrix. La forma más sencilla de crear un objeto Matrix para el degradado es usar el método `createGradientBox()` de la clase Matrix.

Definición de un objeto Matrix para su utilización con un degradado

Los métodos `beginGradientFill()` y `lineGradientStyle()` de la clase flash.display.Graphics se utilizan para definir los degradados que se utilizarán en las formas. Cuando se define un degradado, se proporciona una matriz como uno de los parámetros de estos métodos.

La forma más fácil de definir la matriz consiste en utilizar el método `createGradientBox()` de la clase Matrix, que crea una matriz que se usa para definir el degradado. La escala, la rotación y la posición del degradado se definen con los parámetros que se pasan al método `createGradientBox()`. El método `createGradientBox()` acepta los siguientes parámetros:

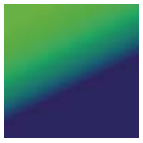
- Anchura del cuadro del degradado: la anchura (en píxeles) en que se extenderá el degradado
- Altura del cuadro del degradado: la altura (en píxeles) sobre la que se extenderá el degradado
- Rotación del cuadro de degradado: la rotación (en radianes) que se aplicará al degradado
- Traslación horizontal: el desplazamiento horizontal (en píxeles) que se aplicará al degradado
- Traslación vertical: el desplazamiento vertical (en píxeles) que se aplicará al degradado

Por ejemplo, en el caso de un degradado con las características siguientes:

- `GradientType.LINEAR`
- Dos colores, verde y azul, con el conjunto `ratios` establecido en `[0, 255]`
- `SpreadMethod.PAD`
- `InterpolationMethod.LINEAR_RGB`

En los ejemplos siguientes, se muestran los degradados en los que el parámetro `rotation` del método `createGradientBox()` difiere del modo indicado, pero el resto de valores no cambia:

`width = 100;`
`height = 100;`
`rotation = 0;`
`tx = 0;`
`ty = 0;`
	

`width = 100;`
`height = 100;`
`rotation = Math.PI/4; // 45°`
`tx = 0;`
`ty = 0;`
	

`width = 100;`
`height = 100;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 0;`
	

En el ejemplo siguiente se muestran los efectos sobre un degradado lineal de verde a azul, en el que los parámetros `rotation`, `tx` y `ty` del método `createGradientBox()` difieren del modo indicado, pero el resto de valores no cambia:

`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 0;`
`ty = 0;`
	

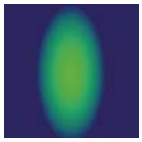
`width = 50;`
`height = 100;`
`rotation = 0`
`tx = 50;`
`ty = 0;`
	

`width = 100;`
`height = 50;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 0;`
	

`width = 100;`
`height = 50;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 50;`
	

Los parámetros `width`, `height`, `tx` y `ty` del método `createGradientBox()` también afectan al tamaño y la posición de un relleno con degradado *radial*, tal como se muestra en el ejemplo siguiente:

```
width = 50;
height = 100;
rotation = 0;
tx = 25;
ty = 0;
```



En el siguiente código se muestra el último degradado radial ilustrado:

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                           colors,
                           alphas,
                           ratios,
                           matrix,
                           spreadMethod,
                           interp,
                           focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

No hay que olvidar que la anchura y altura del relleno degradado vienen dadas por la anchura y la altura de la matriz de degradado en lugar de por la anchura o la altura dibujadas usando el objeto Graphics. Al dibujar con el objeto Graphics, se dibuja lo que existe en esas coordenadas en la matriz de degradado. Incluso si se usa uno de los métodos de forma de un objeto Graphics, como `drawRect()`, el degradado no se estira hasta adaptarse al tamaño de la forma dibujada, sino que el tamaño del degradado debe especificarse en la propia matriz de degradado.

El siguiente ejemplo ilustra la diferencia visual entre las dimensiones de la matriz de degradado y las dimensiones del dibujo:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1,
1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
myShape.graphics.drawRect(0, 50, 100, 40);
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

Este código dibuja tres degradados con el mismo estilo de relleno especificado una distribución idéntica de rojo, verde y azul. Los degradados se dibujan utilizando el método `drawRect()` con anchuras de 50, 100 y 150 píxeles respectivamente. La matriz de degradado especificada en el método `beginGradientFill()` se crea con una anchura de 100 píxeles. Esto quiere decir que el primer degradado abarcará sólo la mitad del espectro del degradado, el segundo lo englobará totalmente y el tercero lo incluirá por completo y además tendrá 50 píxeles más de azul extendiéndose hacia la derecha.

El método `lineGradientStyle()` funciona de forma similar a `beginGradientFill()` excepto en que, además de definir el degradado, es necesario especificar el grosor del trazo utilizando el método `lineStyle()` antes de dibujar. El siguiente código dibuja un cuadrado con un trazo degradado rojo, verde y azul:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1,
1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

Para más información sobre la clase Matrix, consulte "Utilización de objetos Matrix" en la página 358.

Utilización de la clase Math con los métodos de dibujo

Los objetos Graphics pueden dibujar círculos y cuadrados, aunque también pueden trazar formas más complejas, en especial cuando los métodos de dibujo se utilizan junto con las propiedades y métodos de la clase Math. La clase Math contiene constantes comunes de tipo matemático, como `Math.PI` (aproximadamente 3,14159265...), una constante que expresa la proporción entre el perímetro de un círculo y su diámetro. Asimismo, contiene métodos de funciones trigonométricas, como `Math.sin()`, `Math.cos()` y `Math.tan()` entre otros. Cuando se dibujan formas utilizando estos métodos y constantes se crean efectos visuales de mayor dinamismo, especialmente si se usan junto con mecanismos de repetición o recursión.

En muchos métodos de la clase Math es necesario que las unidades empleadas en las medidas circulares sean radianes en lugar de grados. La conversión entre estos dos tipos de unidades es uno de los usos más frecuentes de la clase Math:

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

El siguiente ejemplo crea una onda de seno y una onda de coseno para poner de manifiesto la diferencia entre los métodos `Math.sin()` y `Math.cos()` para un valor dado.

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
    var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
    var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

    graphics.beginFill(sinWaveColor);
    graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
    graphics.beginFill(cosWaveColor);
    graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

Animación con la API de dibujo

Una de las ventajas de crear contenido con la API de dibujo es que no existe la limitación de colocar el contenido una sola vez. Los elementos dibujados se pueden modificar, conservando y modificando las variables utilizadas para dibujar. Se puede lograr un efecto de animación cambiando las variables y redibujando, ya sea a lo largo de una serie de fotogramas o mediante un temporizador.

Por ejemplo, el siguiente código modifica la pantalla con cada fotograma que pasa (detectando el evento `Event.ENTER_FRAME`), incrementa el número de grados e indica al objeto de Graphics que debe limpiar la pantalla y redibujarla con la posición actualizada.

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);

addEventListener(Event.ENTER_FRAME, doEveryFrame);

function doEveryFrame(event:Event):void
{
    currentDegrees += 4;
    var radians:Number = getRadians(currentDegrees);
    var posX:Number = Math.sin(radians) * radius;
    var posY:Number = Math.cos(radians) * radius;
    satellite.graphics.clear();
    satellite.graphics.beginFill(0);
    satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
return degrees * Math.PI / 180;
}
```

Se puede experimentar modificando los valores de inicialización de las variables `currentDegrees`, `radius` y `satelliteRadius` al comienzo del código para lograr unos resultados completamente diferentes. Por ejemplo, se puede reducir la variable radius o incrementar la variable totalSatellites. Esto no es más que un ejemplo de la forma en la que la API de dibujo puede crear una presentación visual cuya complejidad oculta la sencillez de su creación.

Ejemplo: Generador visual algorítmico

El ejemplo del generador visual algorítmico dibuja dinámicamente en el escenario varios "satélites", o círculos que se mueven en una órbita circular. Algunas de las funciones exploradas son:

- Utilizar la API de dibujo para dibujar una forma básica con apariencias dinámicas
- Conectar la interacción del usuario con las propiedades que se usan en un dibujo
- Producir una animación borrando y redibujando el escenario en cada fotograma

El ejemplo de la subsección anterior animaba un "satelite" solitario mediante el evento `Event.ENTER_FRAME`. En este ejemplo se amplía el ejemplo anterior creando un panel de control con varios controles deslizantes que actualizan inmediatamente la presentación visual de varios satélites. En este ejemplo se formaliza el código en clases externas y se rodea con un bucle el código de creación del satélite, almacenando una referencia a cada satélite en un conjunto `satellites`.

Para obtener los archivos de la aplicación para esta muestra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación se encuentran en la carpeta Samples/AlgorithmicVisualGenerator. Esta carpeta contiene los siguientes archivos:

Archivo	Descripción
AlgorithmicVisualGenerator.fla	El archivo principal de la aplicación en Flash (FLA).
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	La clase que proporciona la funcionalidad principal de la aplicación, incluido el dibujo de satélites en el escenario y la respuesta a los eventos del panel de control para actualizar las variables que afectan al dibujo de satélites.
com/example/programmingas3/algorithmic/ControlPanel.as	Una clase que gestiona la interacción con el usuario mediante varios controles deslizantes y distribuye eventos cuando esto ocurre.
com/example/programmingas3/algorithmic/Satellite.as	Una clase que representa al objeto de visualización que gira en una órbita alrededor de un punto central y contiene propiedades relacionadas con su estado de dibujo actual.

Configuración de los detectores

En primer lugar, la aplicación crea tres detectores. El primero de ellos detecta un evento distribuido desde el panel de control que hace que resulte necesario volver a crear los satélites. El segundo detecta los cambios en el tamaño del escenario del archivo SWF. El tercero detecta el paso de los fotogramas en el archivo SWF y redibuja la pantalla utilizando la función `doEveryFrame()`.

Creación de los satélites

Una vez configurados esos detectores, se llama a la función `build()`. Esta función llama en primer lugar a la función `clear()`, que vacía el conjunto `satellites` y borra todo lo que hubiera dibujado anteriormente en el escenario. Esto es necesario, ya que la función `build()` puede volverse a llamar siempre que el panel de control envíe un evento para ello como, por ejemplo, cuando se modifique la configuración del color. En tal caso es necesario eliminar y volver a crear los satélites.

A continuación, la función crea los satélites, configurando las propiedades iniciales necesarias para su creación, como la variable `position`, que se inicia en una posición aleatoria de la órbita, y la variable `color`, que en este ejemplo no cambia una vez creado el satélite.

A medida que se crea cada satélite, se añade una referencia a ellos en el conjunto `satellites`. Cuando se llame a la función `doEveryFrame()`, se actualizarán todos los satélites de este conjunto.

Actualización de la posición de los satélites

La función `doEveryFrame()` es el núcleo del proceso de animación de la aplicación. En cada fotograma se llama a esta función, a una velocidad igual a la velocidad de fotogramas del archivo SWF. Dado que las variables del dibujo varían ligeramente, se produce el efecto de animación.

En primer lugar, la función borra todo lo que hubiese dibujado previamente y redibuja el fondo. A continuación, el bucle itera para cada contenedor de satélite e incrementa la propiedad `position` de cada satélite y actualiza las propiedades `radius` y `orbitRadius`, que pueden haber cambiado a causa de la interacción del usuario con el panel de control. Finalmente, el satélite se actualiza y se coloca en su nueva posición llamando al método `draw()` de la clase Satellite.

Hay que tener presente que el contador, `i`, sólo se incrementa hasta la variable `visibleSatellites`. Esto se debe a que, si el usuario ha limitado la cantidad de satélites que se muestran en el panel de control, los satélites restantes del bucle no se muestran, sino que se ocultan. Esto ocurre en un bucle que sigue inmediatamente al bucle encargado del dibujo.

Cuando se completa la función doEveryFrame(), el número de `visibleSatellites` se actualiza en posición a lo largo de la pantalla.

Respuesta a la interacción con el usuario

La interacción con el usuario se produce a través del panel de control, que gestiona la clase ControlPanel. Esta clase define un detector, junto con los valores mínimos, máximos y predeterminados individuales de cada control deslizante. Cuando el usuario mueve alguno de los controles deslizantes, se llama a la función `changeSetting()`. Esta función actualiza las propiedades del panel de control. Si el cambio requiere que se redibuje la pantalla, se envía un evento que se gestiona en el archivo principal de la aplicación. Cuando cambia la configuración del panel de control, la función `doEveryFrame()` dibuja cada uno de los satélites con las variables actualizadas.

Personalización avanzada

Este ejemplo constituye tan sólo un esquema básico de la forma en que se pueden generar efectos visuales mediante la API de dibujo. Emplea relativamente pocas líneas de código para crear una experiencia interactiva que, en apariencia, es muy compleja y, aún así, este ejemplo puede ampliarse mediante pequeños cambios. A continuación se exponen algunas ideas:

- La función `doEveryFrame()` puede incrementar el valor del color de los satélites.
- La función `doEveryFrame()` puede reducir o aumentar el radio del satélite a lo largo del tiempo.
- El radio del satélite no tiene por qué ser circular. Se puede usar la clase Math para se muevan siguiendo, por ejemplo, una onda sinusoidal.
- Los satélites pueden emplear un mecanismo de detección de colisiones con otros satélites.

Se puede utilizar la API de dibujo como alternativa para crear efectos visuales en el entorno de edición de Flash, dibujando formas básicas en tiempo de ejecución. Asimismo, se puede usar para crear efectos visuales de una variedad y extensión imposibles de alcanzar manualmente. Mediante la API de dibujo y un poco de matemáticas, el autor puede dar vida a un sinnúmero de creaciones sorprendentes.

Utilización avanzada de la API de dibujo

Introducción a la utilización de la API de dibujo avanzada

Flash Player 10 y Adobe AIR 1.5 incorporan compatibilidad para un conjunto avanzado de funciones de dibujo. Las mejoras de la API de dibujo amplían los métodos de dibujo de las versiones anteriores, por lo que se pueden establecer conjuntos de datos para generar formas, modificarlas en tiempo de ejecución y crear efectos tridimensionales. Las mejoras de la API de dibujo consolidan los métodos existentes como comandos alternativos. Estos comandos aprovechan los conjuntos vectoriales y las clases de enumeración para ofrecer conjuntos de datos para métodos de dibujo. El uso de conjuntos vectoriales permite que las formas más complejas se representen con rapidez y que los desarrolladores puedan modificar los valores del conjunto mediante programación para la representación de formas dinámicas en tiempo de ejecución.

Las funciones de dibujo incluidas en Flash Player 10 se describen en las siguientes secciones: "Trazados de dibujo" en la página 343, "Definición de reglas de trazo" en la página 345, "Utilización de clases de datos gráficos" en la página 347 y "Utilización de drawTriangles()" en la página 350.

Tareas comunes de la API de dibujo avanzada

A continuación se enumeran tareas habituales que se realizan con la API de dibujo avanzado de ActionScript:

- Uso de objetos Vector para almacenar datos para métodos de dibujo.
- Definición de trazados para dibujar formas mediante programación.
- Definición de reglas de dirección de trazado para determinar cómo se rellenan las formas superpuestas.
- Utilización de clases de datos gráficos
- Uso de triángulos y métodos de dibujo para efectos tridimensionales.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en esta sección:

- Vector: conjunto de valores del mismo tipo de datos. Un objeto Vector puede almacenar un conjunto de valores que los métodos de dibujo utilizan para crear líneas y formas con un solo comando. Para obtener más información sobre los objetos Vector, consulte "Conjuntos indexados" en la página 161.
- Trazado: un trazado se compone de uno o varios segmentos rectos o curvos. El principio y el final de cada segmento están marcados con coordenadas, que funcionan como grapas que mantienen un cable en su lugar. Un trazado puede estar cerrado (por ejemplo, un círculo), o abierto, con marcados puntos finales (por ejemplo, una línea ondulante).
- Dirección de trazado: la dirección de un trazado interpretada por el procesador; positiva (en el sentido de las agujas del reloj) o negativa (en el sentido contrario a las agujas del reloj).
- GraphicsStroke: clase para definir el estilo de línea. Aunque el "trazo" no se incluye entre las mejoras de la API de dibujo, el uso de una clase para designar un estilo de línea con su propia propiedad de relleno sí supone una mejora. Se puede ajustar dinámicamente un estilo de línea utilizando la clase GraphicsStroke.
- Objetos Fill: objetos creados utilizando clases de visualización como flash.display.GraphicsBitmapFill y flash.display.GraphicsGradientFill, que se transmiten al comando de dibujo `Graphics.drawGraphicsData()`. Los objetos Fill y los comandos de dibujo mejorados introducen un enfoque de programación más orientado a objetos en la réplica de `Graphics.beginBitmapFill()` y `Graphics.beginGradientFill()`.

Trazados de dibujo

La sección sobre dibujo de líneas y curvas (consulte "Dibujo de líneas y curvas" en la página 331) introdujo los comandos para dibujar una sola línea (`Graphics.lineTo()`) o curva (`Graphics.curveTo()`) y posteriormente mover la línea a otro punto (`Graphics.moveTo()`) para realizar una forma. Flash Player 10 y Adobe AIR 1.5 incorporan compatibilidad con las mejoras de la API de dibujo de ActionScript, como `Graphics.drawPath()`.y `Graphics.drawTriangles()`, que utilizan los comandos de dibujo existentes como parámetros. Por lo tanto, una serie de comandos `Graphics.lineTo()`, `Graphics.curveTo()` o `Graphics.moveTo()` se ejecutan en una sola sentencia.

Dos de las mejoras de la API de dibujo permiten que `Graphics.drawPath()` y `Graphics.drawTriangles()` consoliden comandos existentes:

- Clase de enumeración GraphicsPathCommand: la clase GraphicsPathCommand asocia varios comandos de dibujo con valores constantes. Se usa una serie de estos valores como parámetros para el método `Graphics.drawPath()`. Posteriormente con un solo comando puede representar una forma completa o varias formas. También puede modificar dinámicamente los valores transmitidos a estos métodos para modificar una forma existente.
- Conjuntos vectoriales: contiene una serie de valores de un tipo de datos específico. Por lo tanto, se almacena una serie de constantes GraphicsPathCommand en un objeto Vector y varias coordenadas en otro objeto Vector. `Graphics.drawPath()` o `Graphics.drawTriangles()` asigna estos valores de forma conjunta para generar un trazado de dibujo o forma.

Ya no es necesario separar comandos para cada segmento de una forma. Por ejemplo, el método `Graphics.drawPath()` consolida `Graphics.moveTo()`, `Graphics.lineTo()` y `Graphics.curveTo()` en un solo método. En lugar de que los métodos se llamen por separado, se abstraen en identificadores numéricos, tal y como se define en la clase GraphicsPathCommand. Una operación `moveTo()` se expresa mediante un `1`, mientras que la operación `lineTo()` es un `2`. Almacene un conjunto de estos valores en un objeto Vector.<int> object para su uso en el parámetro `commands`. A continuación, cree otro conjunto que contenga coordenadas en un objeto Vector.<Number> para el parámetro `data`. Cada valor GraphicsPathCommand se corresponde con los valores de coordenadas almacenados en el parámetro de datos donde dos números consecutivos definen una ubicación en el espacio de coordenadas de destino.

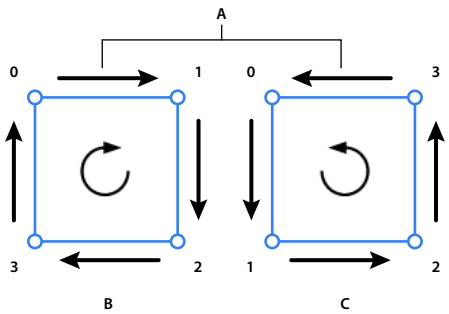
Nota: *los valores en el vector no son objetos Point; el vector es una serie de números donde cada grupo de dos números representa un par de coordenadas x/y.*

El método `Graphics.drawPath()` hace coincidir cada comando con sus respectivos valores de coma (un conjunto de dos o cuatro números) para generar un trazado en el objeto Graphics:

```
package{
import flash.display.*;

public class DrawPathExample extends Sprite {
    public function DrawPathExample(){

    var square_commands:Vector.<int> = new Vector.<int>(5,true);
    square_commands[0] = 1;//moveTo
    square_commands[1] = 2;//lineTo
    square_commands[2] = 2;
    square_commands[3] = 2;
    square_commands[4] = 2;

    var square_coord:Vector.<Number> = new Vector.<Number>(10,true);
    square_coord[0] = 20; //x
    square_coord[1] = 10; //y
    square_coord[2] = 50;
    square_coord[3] = 10;
    square_coord[4] = 50;
    square_coord[5] = 40;
    square_coord[6] = 20;
    square_coord[7] = 40;
    square_coord[8] = 20;
    square_coord[9] = 10;

    graphics.beginFill(0x442266);//set the color
    graphics.drawPath(square_commands, square_coord);
    }
}
}
```

Definición de reglas de trazo

Flash Player 10 y Adobe AIR 1.5 también introducen el concepto de "dirección de trazado". La dirección de trazado es positiva (en el sentido de las agujas del reloj) o negativa (en el sentido contrario a las agujas del reloj). El orden en el que el procesador interpreta las coordenadas que proporciona el vector para el parámetro de datos determina la dirección de trazado.



Dirección de trazado positivo y negativo
A. *Las flechas indican la dirección de trazado* ***B.*** *Dirección de trazado de curva positiva (en el sentido de las agujas del reloj)* ***C.*** *Dirección de trazado de curva negativa (en el sentido contrario a las agujas del reloj)*

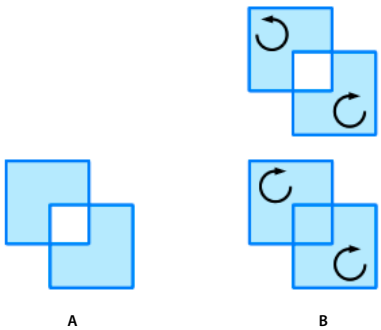
Asimismo, se debe tener en cuenta que el método `Graphics.drawPath()` tiene un tercer parámetro opcional denominado "winding":

```
drawPath(commands:Vector.<int>, data:Vector.<Number>, winding:String = "evenOdd"):void
```

En este contexto, el tercer parámetro es una cadena o una constante que especifica la dirección de trazado o regla de relleno para los trazados de intersección. (Los valores de la constante se definen en la clase GraphicsPathWinding como`GraphicsPathWinding.EVEN_ODD` o `GraphicsPathWinding.NON_ZERO`.) La regla de dirección de trazado es importante cuando los trazados se intersecan.

La regla par-impar es la regla de dirección de trazado estándar y se utiliza en la API de dibujo heredada. La regla par-impar también es la predeterminada para el método `Graphics.drawPath()`. Con la regla de dirección de trazado par-impar, cualquier trazado de intersección alterna entre rellenos abiertos y cerrados. Si dos cuadrados dibujados con el mismo relleno intersecan, el área de intersección se rellena. Generalmente, las áreas adyacentes no están ambas rellenas ni vacías.

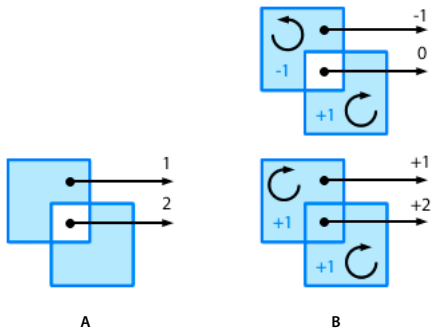
Por otra parte, la regla distinta a cero depende de la dirección de trazado(dirección de dibujo) para determinar si se rellenan las áreas definidas mediante trazados de intersección. Cuando los trazados de dirección opuesta se intersecan, el área definida no se rellena, al igual que con la regla par-impar. Para los trazados con la misma dirección, el área que quedaría vacía se rellena:



Reglas de dirección de trazado para áreas de intersección
***A.** Regla de dirección de trazado par-impar* ***B.** Regla de dirección de trazado distinta a cero*

Nombres de reglas de dirección de trazado

Los nombres hacen referencia a una regla más específica que define el modo en que se administran estos rellenos. Los trazados de curva positiva se asignan como un valor de +1; los trazados de curva negativa se asignan con un valor de -1. Comenzando desde un punto en un área cerrada de una forma, dibuje una línea desde ese punto y prolónguela indefinidamente. El número de veces que la línea cruza un trazado y los valores combinados de esos trazados se utilizan para determinar el relleno. Para la dirección de trazado par-impar, se utiliza el recuento de veces que la línea cruza un trazado. Si el recuento es impar, el área se rellena. Para los recuentos pares, el área no se rellena. Para la dirección de trazado distinta a cero, se utilizan los valores asignados a los trazados. Cuando los valores combinados del trazado son distintos a 0, el área se rellena. Si el valor combinado es 0, el área no se rellena.



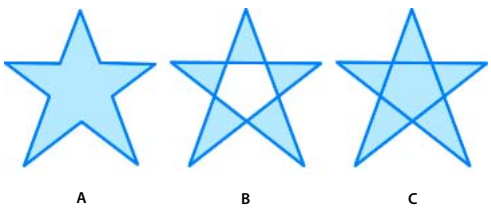
Rellenos y recuentos de la regla de dirección de trazado
***A.** Regla de dirección de trazado par-impar **B.** Regla de dirección de trazado distinta a cero*

Uso de reglas de dirección de trazado

Estas reglas de relleno son complicadas, pero en algunas situaciones resultan necesarias. Pongamos como ejemplo el dibujo de una forma de estrella. Con la regla par-impar estándar, la forma necesitaría diez líneas diferentes. Con la regla de dirección de trazado distinta a cero, estas diez líneas se reducen a cinco. A continuación se muestra la secuencia de ActionScript para una estrella con cinco líneas y una regla de dirección de trazado distinta a cero:

```
fill.graphics.beginFill(0x60A0FF);graphics.drawPath( Vector.<int>([1,2,2,2,2]),
Vector.<Number>([66,10, 23,127, 122,50, 10,49, 109,127]),   GraphicsPathWinding.NON_ZERO);
```

Y a continuación se muestra la forma de estrella:



Forma de estrella con distintas reglas de dirección de trazado
***A.** 10 líneas (par-impar) **B.** 5 líneas (par-impar) **C.** 5 líneas distintas a cero*

A medida que las imágenes se animan o se utilizan como texturas en objetos tridimensionales y se superponen, las reglas de dirección de trazado cobran mayor importancia.

Utilización de clases de datos gráficos

Flash Player 10 y Adobe AIR 1.5 incorporan una colección de clases en el paquete flash.display del tipo IGraphicsData (una interfaz que implementan todas las clases). las clases que implementan la interfaz IGraphicsData sirven como contenedores de datos para los métodos de la API de dibujo.

Las siguientes clases implementan la interfaz IGraphicsData:

- GraphicsBitmapFill
- GraphicsEndFill
- GraphicsGradientFill
- GraphicsPath

- GraphicsShaderFill
- GraphicsSolidFill
- GraphicsStroke
- GraphicsTrianglePath

Con estas clases, puede almacenar dibujos completos en un conjunto de objetos vectoriales de tipo IGraphicsData (Vector.<IGraphicsData>) que se puede reutilizar como el origen de datos para otras instancias de forma o para almacenar información de dibujo para su uso posterior.

Se debe tener en cuenta que se dispone de varias clases de relleno para cada estilo de relleno, pero sólo de una clase de trazo. ActionScript dispone únicamente de la clase de trazo IGraphicsData, ya que esta clase utiliza las clases de relleno para definir su estilo. Por lo tanto, cada trazo es realmente la clase de trazo y una clase de relleno. De lo contrario, la API para estas clases de datos gráficos reflejan los métodos que representan en la clase flash.display.Graphics:

Método Graphics	**Clase de datos**
beginBitmapFill()	GraphicsBitmapFill
beginFill()	GraphicsSolidFill
beginGradientFill()	GraphicsGradientFill
beginShaderFill()	GraphicsShaderFill
lineBitmapStyle()	GraphicsStroke + GraphicsBitmapFill
lineGradientStyle()	GraphicsStroke + GraphicsGradientFill
lineShaderStyle()	GraphicsStroke + GraphicsShaderFill
lineStyle()	GraphicsStroke + GraphicsSolidFill
moveTo() lineTo() curveTo() drawPath()	GraphicsPath
drawTriangles()	GraphicsTrianglePath

Asimismo, la clase GraphicsPath tiene sus propios métodos de utilidad `GraphicsPath.moveTo()`, `GraphicsPath.lineTo()`, `GraphicsPath.curveTo()`, `GraphicsPath.wideLineTo()` y `GraphicsPath.wideMoveTo()` para definir fácilmente los comandos para una instancia de GraphicsPath. Estos métodos de utilidad facilitan la definición o actualización de los comandos y valores de datos directamente.

Una vez que se dispone de una colección de instancias de IGraphicsData, utilice `Graphics.drawGraphicsData()` para representar los gráficos. El método `Graphics.drawGraphicsData()` ejecuta un vector de las instancias de IGraphicsData mediante la API de dibujo en orden secuencial:

```
// stroke object
var stroke:GraphicsStroke = new GraphicsStroke(3);
stroke.joints = JointStyle.MITER;
stroke.fill = new GraphicsSolidFill(0x102020);// solid stroke

// fill object
var fill:GraphicsGradientFill = new GraphicsGradientFill();
fill.colors = [0x0000FF, 0xEEFFEE];
fill.matrix = new Matrix();
fill.matrix.createGradientBox(70,70, Math.PI/2);
// path object
var path:GraphicsPath = new GraphicsPath(new Vector.<int>(), new Vector.<Number>());
path.commands.push(1,2,2);
path.data.push(125,0, 50,100, 175,0);

// combine objects for complete drawing
var drawing:Vector.<IGraphicsData> = new Vector.<IGraphicsData>();
drawing.push(stroke, fill, path);

// draw the drawing
graphics.drawGraphicsData(drawing);
```

Con la modificación de un valor en el trazado utilizado por el dibujo en el ejemplo, la forma se puede volver a dibujar varias veces para una imagen más compleja:

```
// draw the drawing multiple times
// change one value to modify each variation
graphics.drawGraphicsData(drawing);
path.data[2] += 200;
graphics.drawGraphicsData(drawing);
path.data[2] -= 150;
graphics.drawGraphicsData(drawing);
path.data[2] += 100;
graphics.drawGraphicsData(drawing);
path.data[2] -= 50;graphicsS.drawGraphicsData(drawing);
```

Aunque los objetos IGraphicsData pueden definir estilos de trazo y de relleno, éstos no constituyen un requisito. Es decir, los métodos de la clase Graphics se pueden utilizar para definir estilos, mientras que los objetos IGraphicsData se pueden emplear para dibujar una colección de trazados guardados, o viceversa.

Nota: *utilice el método* `Graphics.clear()` *para borrar un dibujo anterior antes de comenzar uno nuevo, a no ser que se trate de una adición al dibujo original, tal y como se muestra en el ejemplo anterior. Conforme cambia una sola parte de un trazado o colección de objetos IGraphicsData, vuelva a dibujar todo el dibujo para ver los cambios.*

Cuando se utilizan clases de datos gráficos, el relleno se representa siempre que se dibujan tres o más puntos, ya que la figura se encuentra cerrada en este punto. Aunque se cierre el relleno, el trazo no lo hace y este comportamiento es diferente al uso de varios comandos `Graphics.lineTo()` o `Graphics.moveTo()`.

Utilización de drawTriangles()

Otro método avanzado incorporado en Flash Player 10 y Adobe AIR 1.5, `Graphics.drawTriangles()` es similar al método `Graphics.drawPath()`. El método `Graphics.drawTriangles()` también utiliza un objeto Vector.<Number> para especificar ubicaciones de punto para dibujar un trazado.

Sin embargo, el principal objetivo del método `Graphics.drawTriangles()` es facilitar efectos tridimensionales en ActionScript. Para obtener más información sobre el uso de `Graphics.drawTriangles()` para generar efectos tridimensionales, consulte "Utilización de triángulos para efectos 3D" en la página 530.

Capítulo 15: Trabajo con la geometría

El paquete flash.geom contiene clases que definen objetos geométricos, como puntos, rectángulos y matrices de transformación. Estas clases se usan para definir las propiedades de los objetos que se aplican en otras clases.

Fundamentos de geometría

Introducción a la utilización de la geometría

Es posible que la geometría sea una de esas asignaturas que la gente intenta aprobar en la escuela y apenas recuerda. Sin embargo, unos conocimientos básicos de geometría pueden ser una eficaz herramienta en ActionScript.

El paquete flash.geom contiene clases que definen objetos geométricos, como puntos, rectángulos y matrices de transformación. Estas clases no siempre proporcionan funcionalidad; sin embargo, se utilizan para definir las propiedades de los objetos que se usan en otras clases.

Todas las clases de geometría se basan en la idea de que las ubicaciones de la pantalla se representan como un plano bidimensional. La pantalla se trata como un gráfico plano con un eje horizontal (x) y un eje vertical (y). Cualquier ubicación (o *punto*) de la pantalla se puede representar como un par de valores x e y (las *coordenadas* de dicha ubicación).

Cada objeto de visualización, incluido el escenario, tiene su propio *espacio de coordenadas*; se trata, básicamente, de un gráfico para representar las ubicaciones de objetos de visualización secundarios, dibujos, etc. Normalmente, el *origen* (el lugar con las coordenadas 0, 0 en el que convergen los ejes x e y) se coloca en la esquina superior izquierda del objeto de visualización. Aunque esto siempre es válido en el caso del escenario, no lo es necesariamente para otros objetos de visualización. Al igual que en los sistemas de coordenadas bidimensionales, los valores del eje x aumentan hacia la derecha y disminuyen hacia la izquierda; en las ubicaciones situadas a la izquierda del origen, la coordenada x es negativa. No obstante, al contrario que los sistemas de coordenadas tradicionales, en ActionScript los valores del eje y aumentan en la parte inferior de la pantalla y disminuyen en la parte superior (los valores por encima del origen tienen una coordenada y negativa). Puesto que la esquina superior izquierda del escenario es el origen de su espacio de coordenadas, los objetos del escenario tendrán una coordenada x mayor que 0 y menor que la anchura del escenario. Asimismo, tendrán una coordenada y mayor que 0 y menor que la altura del escenario.

Se pueden utilizar instancias de la clase Point para representar puntos individuales de un espacio de coordenadas. Además, se puede crear una instancia de Rectangle para representar una región rectangular en un espacio de coordenadas. Los usuarios avanzados pueden utilizar una instancia de Matrix para aplicar varias transformaciones o transformaciones complejas a un objeto de visualización. Muchas transformaciones simples, como la rotación o los cambios de posición y de escala se pueden aplicar directamente a un objeto de visualización mediante las propiedades de dicho objeto. Para obtener más información sobre la aplicación de transformaciones con propiedades de objetos de visualización, consulte “Manipulación de objetos de visualización” en la página 300.

Tareas comunes relacionadas con la geometría

A continuación se enumeran tareas habituales que se realizan con las clases de geometría de ActionScript:

- Calcular la distancia entre dos puntos
- Determinar las coordenadas de un punto en diferentes espacios de coordenadas
- Mover un objeto de visualización variando los valores de ángulo y distancia

- Trabajo con instancias de Rectangle:
  - Cambiar la posición de una instancia de Rectangle
  - Cambiar el tamaño de una instancia de Rectangle
  - Determinar el tamaño combinado o las áreas solapadas de instancias de Rectangle
- Crear objetos Matrix
- Utilizar un objeto Matrix para aplicar transformaciones a un objeto de visualización

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Coordenadas cartesianas: las coordenadas se suelen escribir como un par de números (como 5, 12 o 17, -23). Los dos números son la coordenada x y la coordenada y, respectivamente.
- Espacio de coordenadas; gráfica de coordenadas contenida en un objeto de visualización sobre la que se colocan sus elementos secundarios.
- Origen: punto de un espacio de coordenadas en el que el eje x converge con el eje y. Este punto tiene las coordenadas 0, 0.
- Punto: ubicación individual en un espacio de coordenadas. En el sistema de coordenadas bidimensional utilizado en ActionScript, el punto se define por su ubicación en el eje x y en el eje y (las coordenadas del punto).
- Punto de registro: en un objeto de visualización, el origen (coordenada 0, 0) de su espacio de coordenadas.
- Escala: tamaño de un objeto en relación con su tamaño original. Ajustar la escala de un objeto significa cambiar su tamaño estirándolo o encogiéndolo.
- Trasladar: cambiar las coordenadas de un punto de un espacio de coordenadas a otro.
- Transformación: ajuste de una característica visual de un gráfico, como girar el objeto, modificar su escala, sesgar o distorsionar su forma, o bien modificar su color.
- Eje X: eje horizontal del sistema de coordenadas bidimensional que se utiliza en ActionScript.
- Eje Y: eje vertical del sistema de coordenadas bidimensional que se utiliza en ActionScript.

Ejecución de los ejemplos del capítulo

Muchos de los ejemplos de este capítulo ilustran cálculos o cambios de valores; en la mayor parte se incluyen las llamadas apropiadas a la función `trace()` para mostrar el resultado del código. Para probar estos ejemplos, haga lo siguiente:

1. Cree un documento vacío utilizando la herramienta de edición de Flash.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Ejecute el programa seleccionando Control > Probar película.

   El resultado de las funciones `trace()` del código se ve en el panel Salida.

Algunos de los ejemplos del capítulo ilustran la aplicación de transformaciones a objetos de visualización. Para estos ejemplos los resultados se mostrarán de forma visual, no mediante la salida de texto. Para probar los ejemplos de transformación, haga lo siguiente:

1. Cree un documento vacío utilizando la herramienta de edición de Flash.
2. Seleccione un fotograma clave en la línea de tiempo.

3 Abra el panel Acciones y copie el listado de código en el panel Script.

4 Cree una instancia de símbolo de clip de película en el escenario. Por ejemplo, dibuje una forma, selecciónela, elija Modificar > Convertir en símbolo y asigne un nombre al símbolo.

5 Con el clip de película seleccionado en el escenario, asígnele un nombre de instancia en el inspector de propiedades. El nombre debe coincidir con el utilizado para el objeto de visualización en el listado de código de ejemplo; por ejemplo, si el código aplica una transformación a un objeto denominado `myDisplayObject`, debe asignar a la instancia de clip de película el nombre `myDisplayObject`.

6 Ejecute el programa seleccionando Control > Probar película.

Verá en pantalla el resultado de las transformaciones que el código aplica al objeto.

En el capítulo "Prueba de los listados de código de ejemplo del capítulo" en la página 36 se explican de forma más detallada las técnicas para la comprobación de listados de código.

Utilización de objetos Point

Un objeto Point define un par de coordenadas cartesianas. Representa una ubicación en un sistema de coordenadas bidimensional, en el que *x* representa el eje horizontal e *y* representa el eje vertical.

Para definir un objeto Point, se establecen sus propiedades `x` e `y` del siguiente modo:

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

Cálculo de la distancia entre dos puntos

Se puede usar el método `distance()` de la clase Point para calcular la distancia entre dos puntos en un espacio de coordenadas. Por ejemplo, el código siguiente calcula la distancia entre los puntos de registro de dos objetos de visualización, `circle1` y `circle2`, en el mismo contenedor de objetos de visualización:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

Traslación de los espacios de coordenadas

Si dos objetos de visualización se encuentran en diferentes contenedores, es posible que se ubiquen en distintos espacios de coordenadas. Se puede usar el método `localToGlobal()` de la clase DisplayObject para trasladar las coordenadas al mismo espacio de coordenadas (global), es decir, el del escenario. Por ejemplo, el código siguiente calcula la distancia entre los puntos de registro de dos objetos de visualización, `circle1` y `circle2`, en diferentes contenedores de objetos de visualización:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle2.x, circle2.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

Asimismo, para calcular la distancia del punto de registro de un objeto de visualización denominado `target` con respecto a un punto específico del escenario, se puede usar el método `localToGlobal()` de la clase DisplayObject:

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

Desplazamiento de un objeto de visualización con un ángulo y una distancia especificados

Se puede usar el método `polar()` de la clase Point para desplazar un objeto de visualización una distancia específica con un ángulo determinado. Por ejemplo, el código siguiente desplaza el objeto `myDisplayObject` 100 píxeles por 60 grados:

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Otros usos de la clase Point

Los objetos Point se pueden usar con los métodos y las propiedades siguientes:

Clase	Métodos o propiedades	Descripción
DisplayObjectContainer	`areInaccessibleObjectsUnderPoint()getObjectsUnderPoint()`	Se usa para devolver una lista de objetos bajo un punto en un contenedor de objetos de visualización.
BitmapData	`hitTest()`	Se usa para definir el píxel en el objeto BitmapData, así como el punto en el que se busca una zona activa.

BitmapData	`applyFilter()`
	`copyChannel()`
	`merge()`
	`paletteMap()`
	`pixelDissolve()`
	`threshold()`
		Se usa para definir las posiciones de los rectángulos que definen las operaciones.

Matrix	`deltaTransformPoint()` `transformPoint()`	Se usa para definir los puntos para los que se desea aplicar una transformación.

Rectangle	`bottomRight`
	`size`
	`topLeft`
		Se usa para definir estas propiedades.

Utilización de objetos Rectangle

Un objeto Rectangle define un área rectangular. Tiene una posición, definida por las coordenadas *x* e *y* de su esquina superior izquierda, una propiedad `width` y una propiedad `height`. Estas propiedades se pueden definir para un objeto Rectangle mediante una llamada a la función constructora `Rectangle()`, del siguiente modo:

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

Cambio de tamaño y posición de objetos Rectangle

Existen varias formas de cambiar el tamaño y la posición de los objetos Rectangle.

Se puede cambiar la posición del objeto Rectangle directamente modificando sus propiedades `x` e `y`. Esta acción no tiene ningún efecto sobre la anchura o la altura del objeto Rectangle.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

Tal como muestra el código siguiente, si se cambia la propiedad `left` o `top` de un objeto Rectangle, también se modifica la posición de dicho objeto; las propiedades `x` e `y` coinciden, respectivamente, con las propiedades `left` y `top`. Sin embargo, no se modifica la posición de la esquina inferior izquierda del objeto Rectangle y cambia su tamaño.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=20, y=30, w=80, h=20)
```

Asimismo, tal como se muestra en el ejemplo siguiente, si se cambia la propiedad `bottom` o `right` del objeto Rectangle, no se verá modificada la posición de su esquina superior izquierda y el tamaño cambiará:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

También se puede cambiar la posición de un objeto Rectangle con el método `offset()`, del siguiente modo:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

El método `offsetPt()` funciona de un modo similar, con la excepción de que utiliza un objeto Point como parámetro, en lugar de valores de desplazamiento *x* e *y*.

También se puede cambiar el tamaño de un objeto Rectangle con el método `inflate()`, que incluye dos parámetros, `dx` y `dy`. El parámetro `dx` representa el número de píxeles que los lados izquierdo y derecho del rectángulo se desplazarán con respecto al centro; el parámetro `dy` representa el número de píxeles que las partes superior e inferior del rectángulo se desplazarán con respecto al centro:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

El método `inflatePt()` funciona de un modo similar, con la excepción de que utiliza un objeto Point como parámetro, en lugar de valores `dx` y `dy`.

Búsqueda de uniones e intersecciones de objetos Rectangle

El método `union()` se usa para buscar la región rectangular formada por los límites de dos rectángulos:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

El método `intersection()` se usa para buscar la región rectangular formada por el área solapada de dos rectángulos:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

El método `intersects()` se usa para saber si dos rectángulos presentan un punto de intersección. Asimismo, se puede usar el método `intersects()` para conocer si en una región determinada del escenario hay un objeto de visualización. Por ejemplo, en el código siguiente, se considera que el espacio de coordenadas del contenedor de objetos de visualización que incluye el objeto `circle` es el mismo que el del escenario. En el ejemplo se muestra cómo utilizar el método `intersects()` para determinar si un objeto de visualización, `circle`, crea una intersección con regiones especificadas del escenario, definidas por los objetos Rectangle `target1` y `target2`:

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

Asimismo, se puede utilizar el método `intersects()` para saber si se superponen los rectángulos de delimitación de dos objetos de visualización. Se puede utilizar el método `getRect()` de la clase DisplayObject para incluir el espacio adicional que puedan añadir los trazos de un objeto de visualización a la región de delimitación.

Utilización alternativa de objetos Rectangle

Los objetos Rectangle se utilizan en los métodos y las propiedades siguientes:

Clase	Métodos o propiedades	Descripción
BitmapData	`applyFilter()`, `colorTransform()`, `copyChannel()`, `copyPixels()`, `draw()`, `fillRect()`, `generateFilterRect()`, `getColorBoundsRect()`, `getPixels()`, `merge()`, `paletteMap()`, `pixelDissolve()`, `setPixels()` y `threshold()`	Se usa como el tipo de algunos parámetros para definir una región del objeto BitmapData.
DisplayObject	`getBounds()`, `getRect()`, `scrollRect`, `scale9Grid`	Se usa como el tipo de datos de la propiedad o el tipo de datos que se devuelve.
PrintJob	`addPage()`	Se usa para definir el parámetro `printArea`.
Sprite	`startDrag()`	Se usa para definir el parámetro `bounds`.
TextField	`getCharBoundaries()`	Se usa como el tipo de valor devuelto.
Transform	`pixelBounds`	Se usa como el tipo de datos.

Utilización de objetos Matrix

La clase Matrix representa una matriz de transformación que determina cómo asignar puntos de un espacio de coordenadas a otro. Es posible realizar varias transformaciones gráficas en un objeto de visualización mediante la definición de las propiedades de un objeto Matrix y aplicar dicho objeto Matrix a la propiedad `matrix` de un objeto Transform. Por último, se puede aplicar dicho objeto Transform como la propiedad `transform` del objeto de visualización. Estas funciones de transformación son la traslación (cambio de posición *x* e *y*), giro, cambio de escala y sesgo.

Definición de objetos Matrix

Aunque se puede definir una matriz ajustando directamente las propiedades (`a`, `b`, `c`, `d`, `tx`, `ty`) de un objeto Matrix, resulta más fácil utilizar el método `createBox()`. Este método incluye parámetros que permiten definir directamente los efectos de ajuste de escala, rotación y traslación del objeto Matrix resultante. Por ejemplo, el código siguiente crea un objeto Matrix que ajusta la escala de un objeto con un factor de 2 horizontalmente y 3 verticalmente, lo gira 45 grados y lo mueve (es decir, lo traslada) 10 píxeles a la derecha y 20 píxeles hacia abajo:

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

También se pueden ajustar los efectos de escala, rotación y traslación de un objeto Matrix con los métodos `scale()`, `rotate()` y `translate()`. Estos métodos se combinan con los valores del objeto Matrix existente. Por ejemplo, en el código siguiente se establece un objeto Matrix que ajusta la escala de un objeto con un factor de 4 y lo gira 60 grados, ya que los métodos `scale()` y `rotate()` se llaman dos veces:

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

Para aplicar una transformación de sesgo a un objeto Matrix, debe ajustarse su propiedad `b` o `c`. Al ajustar la propiedad `b`, se sesga la matriz verticalmente y al ajustar la propiedad `c`, se sesga la matriz horizontalmente. En el código siguiente, se sesga el objeto Matrix `myMatrix` verticalmente con un factor de 2:

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

Se puede aplicar una transformación de objeto Matrix a la propiedad `transform` de un objeto de visualización. Por ejemplo, el código siguiente aplica una transformación de matriz a un objeto de visualización denominado `myDisplayObject`:

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

La primera línea define un objeto Matrix en la matriz de transformación existente que utiliza el objeto de visualización `myDisplayObject` (la propiedad `matrix` de la propiedad `transformation` del objeto de visualización `myDisplayObject`). De este modo, los métodos de la clase Matrix que se llamen tendrán un efecto acumulativo sobre la posición, la escala y la rotación del objeto de visualización.

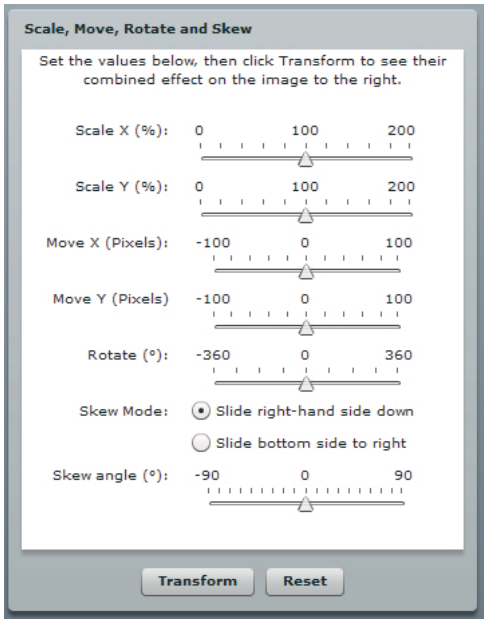
Nota: *la clase ColorTransform también está incluida en el paquete flash.geometry. Esta clase se utiliza para establecer la propiedad* `colorTransform` *de un objeto Transform. Puesto que no aplica ningún tipo de transformación geométrica, no se describe en este capítulo. Para obtener más información, consulte la clase ColorTransform en Referencia del lenguaje y componentes ActionScript 3.0.*

Ejemplo: Aplicación de una transformación de matriz a un objeto de visualización

La aplicación de ejemplo DisplayObjectTransformer muestra varias funciones de la utilización de la clase Matrix para transformar un objeto de visualización, entre las que se incluyen:

- Girar el objeto de visualización
- Ajustar la escala del objeto de visualización
- Trasladar el objeto de visualización (cambiar su posición)
- Sesgar el objeto de visualización

La aplicación proporciona una interfaz para ajustar los parámetros de la transformación de matriz del modo siguiente:



Cuando el usuario hace clic en el botón Transformar, la aplicación aplica la transformación correspondiente.

El objeto de visualización original, y el mismo objeto con una rotación de -45 y un ajuste de escala del 50%

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación DisplayObjectTransformer se encuentran en la carpeta Samples/DisplayObjectTransformer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
DisplayObjectTransformer.mxml o DisplayObjectTransformer.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/geometry/MatrixTransformer.as	Una clase que contiene métodos para aplicar transformaciones de matriz.
img/	Un directorio que contiene archivos de imagen de ejemplo que utiliza la aplicación.

Definición de la clase MatrixTransformer

La clase MatrixTransformer incluye métodos estáticos que aplican transformaciones geométricas de objetos Matrix.

Método transform()

El método `transform()` incluye parámetros para cada uno de los valores siguientes:

- `sourceMatrix`: matriz de entrada que es transformada por el método.
- `xScale` e `yScale`: factor de escala *x* e *y*.
- `dx` y `dy`: magnitudes de traslación *x* e *y*, representadas en píxeles.
- `rotation`: magnitud de rotación, expresada en grados.
- `skew`: factor de sesgo, como porcentaje.
- `skewType`: dirección del sesgo, ya sea `"right"` (derecha) o `"left"` (izquierda).

El valor devuelto es la matriz resultante.

El método `transform()` llama a los siguientes métodos estáticos de la clase:

- `skew()`
- `scale()`
- `translate()`
- `rotate()`

Cada uno devuelve la matriz de origen con la transformación aplicada.

Método skew()

El método `skew()` sesga la matiz ajustando las propiedades `b` y `c` de la misma. Un parámetro opcional, `unit`, determina las unidades que se utilizan para definir el ángulo de sesgo y, si es necesario, el método convierte el valor `angle` en radianes:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

Se crea y se ajusta un objeto `skewMatrix` para aplicar la transformación de sesgo. Inicialmente, es la matriz de identidad:

```
var skewMatrix:Matrix = new Matrix();
```

El parámetro `skewSide` determina el lado en el que se aplica el sesgo. Si se establece en `"right"`, el código siguiente establece la propiedad `b` de la matriz:

```
skewMatrix.b = Math.tan(angle);
```

De lo contrario, el lado inferior se sesga ajustando la propiedad `c` de la matriz, del siguiente modo:

```
skewMatrix.c = Math.tan(angle);
```

El sesgo resultante se aplica a la matriz existente mediante la concatenación de las dos matrices, como se muestra en el ejemplo siguiente:

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

Método scale()

Como se muestra en el ejemplo, el método `scale()` ajusta primero el factor de escala si se proporciona como porcentaje, y luego utiliza el método `scale()` del objeto de matriz:

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

Método translate()

El método `translate()` aplica simplemente los factores de traslación `dx` y `dy` llamando al método `translate()` del objeto de matriz:

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

Método rotate()

El método `rotate()` convierte el factor de rotación de entrada en radianes (si se proporciona en grados o gradientes), y luego llama al método `rotate()` del objeto de matriz:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

Llamada al método MatrixTransformer.transform() desde la aplicación

La aplicación contiene una interfaz de usuario para obtener los parámetros de transformación del usuario. A continuación, pasa dichos parámetros, junto con la propiedad `matrix` de la propiedad `transform` del objeto de visualización, al método `Matrix.transform()`:

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
    xScaleSlider.value,
    yScaleSlider.value,
    dxSlider.value,
    dySlider.value,
    rotationSlider.value,
    skewSlider.value,
    skewSide );
```

La aplicación aplica el valor devuelto a la propiedad `matrix` de la propiedad `transform` del objeto de visualización, con lo que se activa la transformación:

```
img.content.transform.matrix = tempMatrix;
```

Capítulo 16: Aplicación de filtros a objetos de visualización

Históricamente, la aplicación de efectos de filtro a imágenes de mapa de bits ha sido un campo reservado al software especializado de edición de imágenes, como Adobe Photoshop® y Adobe Fireworks®. ActionScript 3.0 incluye el paquete flash.filters, que contiene una serie de clases de filtros de efectos para mapas de bits con el que los desarrolladores pueden aplicar mediante programación filtros a mapas de bits y objetos de visualización a fin de lograr muchos de los efectos disponibles en las aplicaciones de manipulación de gráficos.

Fundamentos de la aplicación de filtros a los objetos de visualización

Introducción a la aplicación de filtros a los objetos de visualización

Una de las formas de realizar la apariencia de una aplicación es añadir efectos gráficos sencillos, como proyectar una sombra tras una foto para lograr una ilusión de tridimensionalidad o un brillo en torno a un botón con objeto de mostrar que se encuentra activo. ActionScript 3.0 incluye nueve filtros que se pueden aplicar a cualquier objeto de visualización o instancia de BitmapData. Entre ellos hay desde filtros básicos, como los de sombra e iluminado, hasta filtros complejos para crear efectos diversos, como los de mapa de desplazamiento y convolución.

Tareas comunes de aplicación de filtros

A continuación se enumera una serie de tareas que se suelen realizar mediante los filtros de ActionScript:

- Crear un filtro
- Aplicar un filtro a un objeto de visualización
- Eliminar un filtro de un objeto de visualización
- Aplicar un filtro a los datos de imagen de una instancia de BitmapData
- Eliminar filtros de un objeto
- Crear varios efectos de filtro, como los de iluminado, desenfoque, sombra, nitidez, desplazamiento, detección de bordes, relieve y de otro tipo

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Bisel: borde creado iluminando los píxeles de dos de los lados y oscureciendo al mismo tiempo los dos lados opuestos, con lo que se crea un efecto de borde tridimensional utilizado normalmente para botones alzados o cincelados y gráficos similares.
- Convolución: distorsión de los píxeles de una imagen obtenida combinando el valor de cada píxel con los valores de algunos o todos sus vecinos en diferentes proporciones.
- Desplazamiento: movimiento, o deslizamiento, de los píxeles de una imagen a una nueva posición.

- Matriz: cuadrícula de números utilizada para realizar determinados cálculos matemáticos aplicando los números de esta cuadrícula a distintos valores y combinando luego los resultados.

Ejecución de los ejemplos del capítulo

A medida que progrese en el estudio de este capítulo, podría desear probar los listados de código de ejemplo proporcionados. Como este capítulo se centra en la creación y manipulación de contenido visual, para probar el código hay que ejecutarlo y ver los resultados en el archivo SWF creado. Casi todos los ejemplos crean contenido mediante la API de dibujo o cargan imágenes a las que se aplican filtros.

Para probar el código de este capítulo:

1. Cree un documento vacío utilizando la herramienta de edición de Flash.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el código en el panel Script.
4. Ejecute el programa seleccionando Control > Probar película.

   Puede ver el resultado del código en el archivo SWF creado.

Casi todos los ejemplos incluyen código que crea una imagen de mapa de bits, por lo que puede probar el código directamente sin necesidad de suministrar contenido de mapa de bits. Como alternativa, se pueden modificar los listados de código para que carguen otras imágenes y las utilicen en lugar de las de los ejemplos.

Creación y aplicación de filtros

Los filtros permiten aplicar toda una serie de efectos a los objetos de visualización y mapas de bits, desde las sombras hasta la creación de biseles y desenfoques. Cada filtro se define como una clase, de manera que al aplicar filtros se crean instancias de objetos de filtro, lo que es exactamente igual que crear cualquier otro objeto. Una vez creada una instancia de un objeto de filtro, se puede aplicar fácilmente a un objeto de visualización utilizando la propiedad `filters` del objeto o, en el caso de un objeto BitmapData, usando el método `applyFilter()`.

Creación de un filtro nuevo

Para crear un nuevo objeto de filtro, basta con llamar al método constructor de la clase de filtro seleccionada. Por ejemplo, para crear un nuevo objeto DropShadowFilter, se usaría el siguiente código:

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

Aunque no se muestra aquí, el constructor `DropShadowFilter()` (como todos los constructores de clases de filtros) acepta varios parámetros opcionales que se pueden utilizar para personalizar la apariencia del efecto de filtro.

Aplicación de un filtro

Una vez construido un objeto de filtro, es posible aplicarlo a un objeto de visualización o a un objeto BitmapData; la forma de aplicarlo depende del objeto al que se vaya a aplicar.

Aplicación de un filtro a un objeto de visualización

Los efectos de filtro se aplican a los objetos de visualización mediante la propiedad `filters`. Dicha propiedad `filters` de los objetos de visualización es una instancia de Array cuyos elementos son los objetos de filtro aplicados al objeto de visualización. Para aplicar un único filtro a un objeto de visualización, se debe crear la instancia de filtro, añadirla a una instancia de Array y asignar ese objeto Array a la propiedad `filters` del objeto de visualización:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// Create a bitmapData object and render it to screen
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// Create a DropShadowFilter instance.
var dropShadow:DropShadowFilter = new DropShadowFilter();

// Create the filters array, adding the filter to the array by passing it as
// a parameter to the Array() constructor.
var filtersArray:Array = new Array(dropShadow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Si se desea asignar varios filtros al objeto, basta con añadir todos los filtros a la instancia de Array antes de asignarla a la propiedad `filters`. Se pueden añadir varios objetos a un elemento Array pasándolos como parámetros a su constructor. Por ejemplo, el siguiente código aplica un filtro de bisel y otro de iluminado al objeto de visualización creado con anterioridad:

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// Create the filters and add them to an array.
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Es posible crear un conjunto que contiene los filtros utilizando el constructor `new Array()` (según se muestra en los ejemplos anteriores) o se puede emplear la sintaxis literal de Array, encerrando los filtros entre corchetes (`[]`). Por ejemplo, esta línea de código:

```
var filters:Array = new Array(dropShadow, blur);
```

hace lo mismo que esta otra:

```
var filters:Array = [dropShadow, blur];
```

Al aplicar varios filtros a objetos de visualización, la aplicación se lleva a cabo de forma acumulativa y secuencial. Por ejemplo, si un conjunto de filtros tiene dos elementos, un filtro de bisel añadido en primer lugar y un filtro de sombra en segundo, el filtro de sombra se aplicará tanto al filtro de bisel como al objeto de visualización. Esto se debe a que el filtro de sombra está en segundo lugar en el conjunto de filtros. Para aplicar filtros de forma no acumulativa, es necesario aplicar cada filtro a una nueva copia del objeto de visualización.

Si sólo se está aplicando un filtro o unos pocos a un objeto de visualización, es posible crear la instancia del filtro y asignarlo al objeto en una única sentencia. Por ejemplo, la siguiente línea de código aplica un filtro de desenfoque a un objeto de visualización llamado `myDisplayObject`:

```
myDisplayObject.filters = [new BlurFilter()];
```

El código anterior crea una instancia de Array utilizando su sintaxis literal (con corchetes), crea una nueva instancia de BlurFilter como elemento de Array y asigna dicho Array a la propiedad `filters` del objeto de visualización llamado `myDisplayObject`.

Eliminación de filtros de un objeto de visualización

Eliminar todos los filtros de un objeto de visualización es tan sencillo como asignar un valor nulo a la propiedad `filters`:

```
myDisplayObject.filters = null;
```

Si se han aplicado varios filtros a un objeto y se desea eliminar solamente uno de ellos, es necesario seguir una serie de pasos para cambiar el conjunto de la propiedad `filters`. Para obtener más información, consulte "Posibles problemas al trabajar con filtros" en la página 366.

Aplicación de un filtro a un objeto BitmapData

Para aplicar un filtro a un objeto BitmapData es necesario utilizar el método `applyFilter()` del objeto BitmapData:

```
var rect:Rectangle = new Rectangle();
var origin:Point = new Point();
myBitmapData.applyFilter(sourceBitmapData, rect, origin, new BlurFilter());
```

El método `applyFilter()` aplica un filtro a un objeto BitmapData de origen, dando lugar a una nueva imagen filtrada. Este método no modifica la imagen original. En vez de eso, el resultado de la aplicación del filtro a la imagen original se almacena en la instancia de BitmapData en la que se llama al método `applyFilter()`.

Funcionamiento de los filtros

Al aplicar un filtro a un objeto de visualización, se crea en caché una copia del objeto original como un mapa de bits transparente.

Una vez que un filtro se ha aplicado a un objeto de visualización, Adobe Flash Player o Adobe® AIR™ almacena en caché el objeto como un mapa de bits mientras éste mantiene una lista de filtros válida. Este mapa de bits de origen se usa luego como imagen original para todas las aplicaciones posteriores de efectos de filtros.

Normalmente, un objeto de visualización contiene dos mapas de bits: uno con el objeto de visualización de origen sin filtrar (original) y otro con la imagen final una vez filtrada. La imagen final es la que se utiliza en representaciones. Mientras el objeto de visualización no cambie, la imagen final no necesita actualización.

Posibles problemas al trabajar con filtros

Existen varias fuentes de confusión o problemas potenciales que hay que tener en cuenta al trabajar con filtros. Todas ellas se describen en las secciones siguientes.

Filtros y caché de mapas de bits

Para aplicar un filtro a un objeto de visualización, debe estar activada la caché de mapa de bits para ese objeto. Al aplicar un filtro a un objeto de visualización cuya propiedad `cacheAsBitmap` tiene el valor `false`, Flash Player o AIR establecen automáticamente el valor de la propiedad `cacheAsBitmap` del objeto como `true`. Si más adelante se quitan todos los filtros del objeto de visualización, Flash Player o AIR restablecen el valor que la propiedad `cacheAsBitmap` tenía con anterioridad.

Cambio de filtros en tiempo de ejecución

Si un objeto de visualización ya tiene aplicados uno o varios filtros, no podrá cambiar este grupo de filtros agregando o eliminando filtros en el conjunto de la propiedad `filters`. En su lugar, para modificar el grupo de filtros aplicados o añadir otros nuevos, debe aplicar los cambios en un conjunto diferente y, a continuación, asignar dicho conjunto a la propiedad filters del objeto de visualización correspondiente a los filtros que se aplicarán al objeto. El modo más sencillo de realizar esta tarea consiste en leer el conjunto de la propiedad `filters` en una variable Array y aplicar las modificaciones en este conjunto temporal. A continuación, se puede volver a asignar este conjunto a la propiedad `filters` del objeto de visualización. En los casos más complejos, es posible que sea necesario mantener un conjunto maestro de filtros diferente. Los cambios se aplicarán a este conjunto maestro para después asignarlo a la propiedad `filters` del objeto de visualización tras cada modificación.

Añadir un filtro adicional

El siguiente código demuestra el proceso de añadir un filtro adicional a un objeto de visualización que ya tiene aplicados uno o varios filtros. Inicialmente se aplica un filtro de iluminado al objeto de visualización llamado `myDisplayObject`; luego, cuando se hace clic en el objeto, se llama a la función `addFilters()`. Mediante esta función se aplican dos filtros más a:`myDisplayObject`:

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
    // Make a copy of the filters array.
    var filtersCopy:Array = myDisplayObject.filters;

    // Make desired changes to the filters (in this case, adding filters).
    filtersCopy.push(new BlurFilter());
    filtersCopy.push(new DropShadowFilter());

    // Apply the changes by reassigning the array to the filters property.
    myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

Eliminación de un filtro de un grupo de filtros

Si un objeto de visualización tiene aplicados varios filtros y desea eliminar uno de ellos y seguir aplicando los demás, cópielos en un conjunto temporal, elimine de éste el filtro que desee y vuelva a asignar el conjunto temporal a la propiedad `filters` del objeto de visualización. En la sección "Recuperación de valores y eliminación de elementos de conjunto" en la página 166 se describen varias formas de eliminar uno o varios elementos de un conjunto.

La forma más sencilla consiste en eliminar el filtro situado al principio del objeto (el último filtro que se le ha aplicado). Se puede utilizar el método `pop()` de la clase Array para eliminar el filtro del conjunto:

```
// Example of removing the top-most filter from a display object
// named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the last element from the Array (the top-most filter).
tempFilters.pop();

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

De igual modo, para eliminar el filtro situado al final (el primero aplicado al objeto) se puede utilizar el mismo código, substituyendo el método `shift()` de la clase Array en lugar del método `pop()`.

Para eliminar un filtro del centro de un conjunto de filtros (suponiendo que el conjunto tenga más de dos filtros), utilice el método `splice()`. Es preciso conocer el índice (la posición en el conjunto) del filtro que se desee eliminar. Por ejemplo, con el siguiente código se elimina el segundo filtro (el del índice 1) de un objeto de visualización:

```
// Example of removing a filter from the middle of a stack of filters
// applied to a display object named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the second filter from the array. It's the item at index 1
// because Array indexes start from 0.
// The first "1" indicates the index of the filter to remove; the
// second "1" indicates how many elements to remove.
tempFilters.splice(1, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

Determinación del índice de un filtro

Es preciso saber qué filtro que se va a eliminar del conjunto, de forma que se pueda identificar el índice correspondiente. Debe conocer (por el modo en que está diseñada la aplicación) o calcular el índice del filtro que se va a eliminar.

El mejor enfoque consiste en diseñar la aplicación de forma que el filtro que se desee eliminar se encuentre siempre en la misma posición en el grupo de filtros. Por ejemplo, si existe un único objeto de visualización que tiene aplicados un filtro de convolución y otro de sombra (en este orden) y desea eliminar este último y mantener el anterior, el filtro se debe situar en una posición conocida (la superior) de forma que se pueda determinar con antelación el método de Array que hay que utilizar (en este caso, `Array.pop()` para eliminar el filtro de sombra).

Si el filtro que desea eliminar siempre es de un tipo específico, pero no se sitúa siempre en la misma posición del grupo de filtros, puede comprobar el tipo de datos de cada filtro del conjunto para determinar cuál hay que eliminar. Por ejemplo, el siguiente código identifica el filtro de iluminado dentro de un grupo de filtros y lo elimina de éste.

```
// Example of removing a glow filter from a set of filters, where the
//filter you want to remove is the only GlowFilter instance applied
// to the filtered object.

var tempFilters:Array = filteredObject.filters;

// Loop through the filters to find the index of the GlowFilter instance.
var glowIndex:int;
var numFilters:int = tempFilters.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (tempFilters[i] is GlowFilter)
    {
        glowIndex = i;
        break;
    }
}

// Remove the glow filter from the array.
tempFilters.splice(glowIndex, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

En un caso más complejo, por ejemplo, cuando el filtro que se va a eliminar se selecciona en tiempo de ejecución, la mejor solución consiste en crear un copia persistente e independiente del conjunto de filtros que sirve como la lista maestra de filtros. Cada vez que modifique el grupo de filtros (añada o elimine un filtro), deberá cambiar la lista maestra y, a continuación, aplicar el conjunto de filtros como la propiedad `filters` del objeto de visualización.

Por ejemplo, en el siguiente listado de código se aplican varios filtros de convolución a un objeto de visualización para crear diferentes efectos visuales y, en un punto posterior de la aplicación, se elimina uno de estos filtros y se mantienen los demás. En este caso, el código conserva una copia maestra del conjunto de filtros, así como una referencia al filtro que se va a eliminar. La identificación y eliminación del filtro específico es un proceso similar al anterior, excepto que en lugar de realizar una copia temporal del conjunto de filtros, se manipula la copia maestra y se aplica después al objeto de visualización.

```
// Example of removing a filter from a set of
// filters, where there may be more than one
// of that type of filter applied to the filtered
// object, and you only want to remove one.

// A master list of filters is stored in a separate,
// persistent Array variable.
var masterFilterList:Array;

// At some point, you store a reference to the filter you
// want to remove.
var filterToRemove:ConvolutionFilter;

// ... assume the filters have been added to masterFilterList,
// which is then assigned as the filteredObject.filters:
filteredObject.filters = masterFilterList;

// ... later, when it's time to remove the filter, this code gets called:

// Loop through the filters to find the index of masterFilterList.
var removeIndex:int = -1;
var numFilters:int = masterFilterList.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (masterFilterList[i] == filterToRemove)
    {
        removeIndex = i;
        break;
    }
}

if (removeIndex >= 0)
{
    // Remove the filter from the array.
    masterFilterList.splice(removeIndex, 1);

    // Apply the new set of filters to the display object.
    filteredObject.filters = masterFilterList;
}
```

En este enfoque (cuando se compara una referencia a un filtro almacenado con los elementos del conjunto de filtros para determinar el filtro que se va a eliminar), se *debe* crear una copia independiente de conjunto de filtros — el código no funciona si se compara la referencia al filtro almacenado con los elementos de un conjunto temporal copiado a partir de la propiedad `filters` del objeto de visualización. El motivo es que al asignar un conjunto a la propiedad `filters`, Flash Player o AIR realizan internamente una copia de cada objeto de filtro del conjunto. Y son estas copias (en lugar de los objetos originales) las que se aplican al objeto de visualización, de forma que al leer la propiedad `filters` en un conjunto temporal, éste contiene referencias a los objetos de filtro copiados en lugar de las referencias a los objetos de filtro originales. En consecuencia, si en el ejemplo anterior se intenta determinar el índice de `filterToRemove` por comparación con los filtros de un conjunto de filtros temporal, no se obtendrá ninguna coincidencia.

Filtros y transformaciones de objetos

Ninguna región filtrada (una sombra, por ejemplo) que se encuentre fuera del recuadro de delimitación rectangular de un objeto de visualización se considera parte de la superficie a efectos de la detección de colisiones (determinar si una instancia se solapa o corta con otra instancia). Dado que los métodos de detección de colisiones de la clase DisplayObject se basan en vectores, no es posible llevar a cabo esta operación sobre un resultado que es un mapa de bits. Por ejemplo, si se aplica un filtro de bisel a una instancia de botón, la detección de colisiones no estará disponible en la parte biselada de la instancia.

Los filtros no son compatibles con el escalado, la rotación ni el sesgo. Si el objeto de visualización al que se le ha aplicado un filtro es escalado (si `scaleX` y `scaleY` no tienen un valor de 100%), el efecto de filtro no se escalará con la instancia. Esto quiere decir que la forma original de la instancia rotará, se escalará o sesgará, pero el filtro no lo hará junto con la instancia.

Se puede animar una instancia con un filtro para crear efectos realistas o anidar instancias y usar la clase BitmapData para animar filtros y así lograr este efecto.

Filtros y objetos de mapas de bits

Al aplicar un filtro a un objeto BitmapData, la propiedad `cacheAsBitmap` toma automáticamente el valor `true`. De este modo, el filtro se aplica en realidad a la copia del objeto en lugar de al original.

Esta copia se coloca a continuación en la visualización principal (sobre el objeto original) tan cerca como sea posible del píxel más cercano. Si los límites del mapa de bits original cambian, la copia del mapa de bits con el filtro se vuelve a crear de nuevo en lugar de estirarla o distorsionarla.

Si se quitan todos los filtros de un objeto de visualización, se restablece el valor que la propiedad `cacheAsBitmap` tenía antes de aplicarse el filtro.

Filtros de visualización disponibles

ActionScript 3.0 incluye diez clases de filtros que se pueden aplicar a objetos de visualización y a objetos BitmapData:

- Filtro de biselado (clase BevelFilter)
- Filtro de desenfoque (clase BlurFilter)
- Filtro de sombra (clase DropShadowFilter)
- Filtro de iluminado (clase GlowFilter)
- Filtro de bisel degradado (clase GradientBevelFilter)
- Filtro de iluminado degradado (clase GradientGlowFilter)
- Filtro de matriz de colores (clase ColorMatrixFilter)
- Filtro de convolución (clase ConvolutionFilter)
- Filtro de mapa de desplazamiento (clase DisplacementMapFilter)
- Filtro de sombreado (clase ShaderFilter)

Los seis primeros son filtros sencillos que se pueden utilizar para crear un efecto específico con posibilidades de personalización limitadas del efecto disponible. Todos ellos se pueden aplicar mediante ActionScript, además de poderse aplicar a objetos en Adobe Flash CS4 Professional desde el panel Filtros. Por lo tanto, incluso si se usa ActionScript para aplicar filtros, se puede utilizar la interfaz visual (si se dispone de la herramienta de edición de Flash) para probar de forma rápida los distintos filtros y configuraciones y, de ese modo, probar la mejor forma de lograr el efecto deseado.

Los últimos cuatro filtros sólo están disponibles en ActionScript. Estos filtros (matriz de colores, convolución, mapa de desplazamiento y sombreado) son mucho más flexibles en los tipos de efectos para cuya creación se pueden utilizar. En lugar de ser optimizados para un único efecto, ofrecen potencia y flexibilidad. Por ejemplo, seleccionando distintos valores para su matriz, el filtro de convolución se puede utilizar para crear efectos de desenfoque, relieve, nitidez, búsqueda de bordes de color, transformaciones y mucho más.

Cada uno de ellos, ya sea sencillo o complejo, se puede personalizar usando sus propiedades. En general, existen dos posibilidades a la hora de definir las propiedades de un filtro. Todos los filtros permiten establecer sus propiedades pasando los valores de los parámetros al constructor del objeto de filtro. Como alternativa, tanto si se definen las propiedades del filtro pasando parámetros como si no, es posible ajustar los filtros más adelante estableciendo valores para las propiedades del objeto de filtro. En la mayor parte de los listados de código, las propiedades se definen directamente para que el ejemplo sea más fácilmente comprensible. No obstante, se suele poder obtener el mismo resultado con menos líneas de código pasando los valores como parámetros en el constructor del objeto de filtro. Para obtener más información sobre cada filtro, sus propiedades y sus parámetros de constructor, consulte el paquete flash.filters en la Referencia del lenguaje y componentes ActionScript 3.0.

Filtro de bisel

La clase BevelFilter permite añadir un borde tridimensional en bisel al objeto filtrado. Este filtro hace que las esquinas o bordes duros del objeto adquieran una apariencia cincelada o biselada.

Las propiedades de la clase BevelFilter permiten personalizar la apariencia del bisel. Se pueden definir los colores de iluminación y sombra, el desenfoque de los bordes del bisel, los ángulos de biselado y la colocación de los bordes del bisel; incluso se puede crear un efecto extractor.

En el siguiente ejemplo se carga una imagen externa y se le aplica un filtro de bisel.

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// Load an image onto the Stage.
var imageLoader:Loader = new Loader();
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);

// Create the bevel filter and set filter properties.
var bevel:BevelFilter = new BevelFilter();

bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// Apply filter to the image.
imageLoader.filters = [bevel];
```

Filtro de desenfoque

La clase BlurFilter difumina, o desenfoca, un objeto de visualización y su contenido. Los efectos de desenfoque resultan útiles para dar la impresión de que un objeto no está bien enfocado o para simular un movimiento rápido, como en el desenfoque por movimiento. Si se define la propiedad `quality` del filtro de desenfoque en un nivel bajo, se puede simular un efecto de lente ligeramente desenfocada. Si la propiedad `quality` se establece en un nivel alto, da lugar a un efecto de desenfoque suave similar a un desenfoque gaussiano.

En el siguiente ejemplo se crea un objeto circular usando el método `drawCircle()` de la clase Graphics y se le aplica un efecto de desenfoque:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// Draw a circle.
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// Add the circle to the display list.
addChild(redDotCutout);

// Apply the blur filter to the rectangle.
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

Filtro de sombra

Las sombras dan la impresión de que existe una fuente de luz independiente situada sobre el objeto de destino. La posición e intensidad de esta fuente de luz se puede modificar para originar toda una gama de distintos efectos de sombra.

La clase DropShadowFilter usa un algoritmo similar al del filtro de desenfoque. La principal diferencia es que el filtro de sombra tiene unas pocas propiedades más que se pueden modificar para simular los distintos atributos de la fuente de luz (como alfa, color, desplazamiento y brillo).

El filtro de sombra también permite aplicar opciones de transformación personalizadas sobre el estilo de la sombra, incluidas la sombra interior o exterior y el modo extractor (también conocido como silueta).

El siguiente código crea una un objeto Sprite cuadrado y le aplica un filtro de sombra:

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// Draw a box.
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// Apply the drop shadow filter to the box.
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// You can also set other properties, such as the shadow color,
// alpha, amount of blur, strength, quality, and options for
// inner shadows and knockout effects.

boxShadow.filters = [shadow];
```

Filtro de iluminado

La clase GlowFilter aplica un efecto de iluminación a los objetos de visualización, de forma que parece que bajo el objeto brilla una luz que lo ilumina suavemente.

El filtro de iluminado es similar al del sombra, pero incluye propiedades para modificar la distancia, el ángulo y el color de la fuente de luz con las que es posible lograr diversos efectos. GlowFilter también dispone de varias opciones para modificar el estilo de la iluminación, entre ellos el brillo interior y exterior y el modo extractor.

El siguiente código crea una cruz usando la clase Sprite y le aplica un filtro de iluminado:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// Create a cross graphic.
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// Apply the glow filter to the cross shape.
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

Filtro de bisel degradado

La clase GradientBevelFilter permite aplicar un efecto de bisel mejorado a los objetos de visualización o a los objetos BitmapData. Al utilizar un degradado de color en el bisel se mejora enormemente su profundidad espacial, dando a los bordes una apariencia en 3D más realista.

El siguiente código crea un objeto rectangular empleando el método `drawRect()` de la clase Shape y le aplica un filtro de bisel degradado.

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// Draw a rectangle.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Apply a gradient bevel to the rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // opposite of 45 degrees
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// Other properties let you set the filter strength and set options
// for inner bevel and knockout effects.

box.filters = [gradientBevel];

// Add the graphic to the display list.
addChild(box);
```

Filtro de iluminado degradado

La clase GradientGlowFilter permite aplicar un efecto de iluminación mejorada a los objetos de visualización o a los objetos BitmapData. Este efecto proporciona un mayor control sobre el color de la iluminación y, de este modo, permite lograr un efecto de brillo más realista. Además, mediante el filtro de iluminado degradado, es posible aplicar un brillo degradado al borde interior, exterior o superior de un objeto.

En el siguiente ejemplo se dibuja un círculo en el escenario y se le aplica un filtro de iluminado degradado. A medida que se mueve el ratón hacia la derecha y abajo, la cantidad de desenfoque aumenta en dirección horizontal y vertical respectivamente. Además, cuando se hace clic en el escenario, la fuerza del desenfoque aumenta.

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// Create a new Shape instance.
var shape:Shape = new Shape();

// Draw the shape.
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// Position the shape on the Stage.
addChild(shape);
shape.x = 100;
shape.y = 100;

// Define a gradient glow.
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;

// Define functions to listen for two events.
function onClick(event:MouseEvent):void
{
    gradientGlow.strength++;
    shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
    gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
    gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
    shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

Ejemplo: Combinación de filtros básicos

En el siguiente ejemplo de código se utilizan varios filtros básicos combinados con un temporizador para crear una simulación animada de la acción repetitiva de un semáforo.

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // opposite of 45 degrees
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// Draw the rectangle background for the traffic light.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Draw the 3 circles for the three lights.
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// Add the graphics to the display list.
addChild(box);
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// Apply a gradient bevel to the traffic light rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance, angleInDegrees,
colors, alphas, ratios, blurX, blurY, strength, quality, type, knockout);
```

```
box.filters = [gradientBevel];

// Create the inner shadow (for lights when off) and glow
// (for lights when on).
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3, 3, 1, 1, true,
false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false, false);
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1, false, false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false, false);

// Set the starting state of the lights (green on, red/yellow off).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// Swap the filters based on the count value.
function trafficControl(event:TimerEvent):void
{
    if (count == 4)
    {
        count = 1;
    }

    switch (count)
    {
        case 1:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [yellowGlow];
            goLight.filters = [innerShadow];
            break;
        case 2:
            stopLight.filters = [redGlow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [innerShadow];
            break;
        case 3:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [greenGlow];
            break;
    }

    count++;
}

// Create a timer to swap the filters at a 3 second interval.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

Filtro de matriz de colores

La clase ColorMatrixFilter se utiliza para manipular los valores de color y alfa del objeto filtrado. Esto permite crear cambios de saturación, rotaciones de matiz (cambio de una paleta de una gama de colores a otra), cambios de luminancia a alfa y otros efectos de manipulación del color usando los valores de un canal de color y aplicándolos potencialmente a otros canales.

Conceptualmente, el filtro recorre de uno en uno los píxeles de la imagen de origen y separa cada uno de ellos en sus componentes rojo, verde, azul y alfa. A continuación, multiplica los valores indicados en la matriz de colores por cada uno de estos valores, sumando los resultados para determinar el valor del color resultante que se mostrará en la pantalla para cada píxel. La propiedad `matrix` del filtro es una conjunto de 20 números que se utiliza para calcular el color final. Para obtener detalles sobre el algoritmo específico utilizado para calcular los valores de color, consulte la entrada que describe la propiedad matrix de la clase ColorMatrixFilter en la Referencia del lenguaje y componentes ActionScript 3.0.

Se puede encontrar más información y ejemplos del filtro de matriz de colores en el artículo (en inglés) "Using Matrices for Transformations, Color Adjustments, and Convolution Effects in Flash" (Utilización de matrices para transformaciones, ajustes de color y efectos de convolución en Flash) en el sitio Web del Centro de desarrollo de Adobe.

Filtro de convolución

La clase ConvolutionFilter se puede utilizar para aplicar una amplia gama de transformaciones gráficas, como efectos de desenfoque, detección de bordes, nitidez, relieve y biselado, a los objetos de visualización o a los objetos BitmapData.

Conceptualmente, el filtro de convolución recorre los píxeles de la imagen de origen de uno en uno y determina su color final basándose en el valor de cada píxel y de los que lo rodean. Una matriz, especificada como un conjunto de valores numéricos, indica la intensidad con la que cada uno de los píxeles circundantes afecta al valor final.

A modo de ejemplo, se puede tomar el tipo de matriz usado con mayor frecuencia, la matriz de tres por tres, que consta de nueve valores:

```
N   N   N
N   P   N
N   N   N
```

Cuando Flash Player o AIR aplican el filtro de convolución a un píxel determinado, éste consulta el valor de color del píxel en sí ("P" en el ejemplo), además de los valores de los píxeles que lo rodean (llamados "N" en el ejemplo). No obstante, al definir los valores de la matriz, se especifica la prioridad que algunos píxeles tendrán a la hora de afectar a la imagen resultante.

Por ejemplo, la siguiente matriz, aplicada mediante un filtro de convolución, dejará la imagen exactamente igual que estaba:

```
0   0   0
0   1   0
0   0   0
```

La razón por la que la imagen no se ve modificada es porque el valor del píxel original tiene una fuerza relativa de 1 para determinar el color final del píxel, mientras que todos los que lo rodean tienen una fuerza relativa de 0, lo que significa que sus colores no afectan a la imagen final.

Del mismo modo, la siguiente matriz hará que los píxeles de una imagen se desplacen un píxel hacia la izquierda:

```
0   0   0
0   0   1
0   0   0
```

Hay que tener en cuenta que, en este caso, el píxel en sí no tiene ningún efecto sobre el valor final del píxel mostrado en esa posición en la imagen final, ya que sólo se usa el valor del píxel de la derecha para determinar el valor resultante de ese píxel.

En ActionScript, la matriz se crea como una combinación de una instancia de Array que contiene los valores y dos propiedades que especifican el número de filas y columnas de la matriz. En el siguiente ejemplo se carga una imagen y, una vez cargada, se le aplica un filtro de convolución utilizando la matriz del listado anterior:

```
// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
    // Create the convolution matrix.
    var matrix:Array = [0, 0, 0,
                                    0, 0, 1,
                                    0, 0, 0];

    var convolution:ConvolutionFilter = new ConvolutionFilter();
    convolution.matrixX = 3;
    convolution.matrixY = 3;
    convolution.matrix = matrix;
    convolution.divisor = 1;

    loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

Algo que no resulta obvio en este código es el efecto que se produce al utilizar valores diferentes a 0 y 1 en la matriz. Por ejemplo, la misma matriz con el número 8 en lugar de 1 en la posición de la derecha, realiza la misma acción (desplazar los píxeles a la izquierda) y, además, afecta a los colores de la imagen, haciéndolos 8 veces más brillantes. Esto se debe a que el color final de los píxeles se calcula multiplicando los valores de matriz por los colores de los píxeles originales, sumando los valores y dividiendo por el valor de la propiedad `divisor` del filtro. Hay que tener en cuenta que en el código de ejemplo, la propiedad `divisor` tiene el valor 1. En general, si se desea que el brillo de los colores permanezca aproximadamente igual que en la imagen original, el divisor debe ser igual a la suma de los valores de la matriz. De este modo, en el caso de una matriz cuyos valores suman 8 y el divisor vale 1, la imagen resultante será aproximadamente 8 veces más brillante que la original.

Si bien el efecto de esta matriz no es demasiado apreciable, es posible utilizar otros valores en la matriz para crear diversos efectos. A continuación se ofrecen varios conjuntos estándar de valores de matriz con los que obtener distintos efectos mediante una matriz de tres por tres:

- Desenfoque básico (divisor 5):

```
0 1 0
1 1 1
0 1 0
```

- Nitidez (divisor 1):

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- Detección de bordes (divisor 1):

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- Efecto de relieve (divisor 1):

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

  Cabe destacar que en la mayor parte de estos efectos el divisor es 1. Esto se debe a que los valores negativos de la matriz, sumandos a los positivos, dan un valor de 1 (0 en el caso de la detección de bordes, pero el valor de la propiedad `divisor` no puede ser 0).

Filtro de mapa de desplazamiento

La clase DisplacementMapFilter usa los valores de los píxeles de un objeto BitmapData (conocido como imagen del mapa de desplazamiento) para llevar a cabo un efecto de desplazamiento sobre un nuevo objeto. La imagen del mapa de desplazamiento suele ser distinta al objeto de visualización o a la instancia de BitmapData a los que se va a aplicar el filtro. En un efecto de desplazamiento, los píxeles de la imagen filtrada se desplazan, es decir, cambian su posición en mayor o menor medida con respecto a la imagen original. Este filtro se puede utilizar para crear un efecto de desplazamiento, combado o moteado.

La ubicación y la cantidad de desplazamiento aplicadas a un píxel determinado vienen dadas por el valor del color de la imagen del mapa de desplazamiento. Cuando se trabaja con el filtro, además de especificar la imagen del mapa, es necesario indicar los siguientes valores para controlar la forma en la que se calcula el desplazamiento a partir de la imagen del mapa:

- Punto del mapa: la posición de la imagen filtrada en la que se aplicará la esquina superior izquierda del filtro de desplazamiento. Se puede usar este valor para aplicar el filtro sólo a una parte de la imagen.
- Componente X: el canal de color de la imagen del mapa que afectará a la posición x de los píxeles.
- Componente Y: el canal de color de la imagen del mapa que afectará a la posición y de los píxeles.
- Escala X: un valor multiplicador que especifica la intensidad del desplazamiento en el eje x.
- Escala Y: un valor multiplicador que especifica la intensidad del desplazamiento en el eje y.
- Modo de filtro: determina la operación que debe realizar Flash Player en los espacios vacíos dejados por los píxeles desplazados. Las opciones, definidas como constantes en la clase DisplacementMapFilterMode, son mostrar los píxeles originales (modo de filtro `IGNORE`), cruzar los píxeles del otro lado de la imagen (modo de filtro `WRAP`, que es el predeterminado), usar el píxel desplazado más cercano (modo de filtro `CLAMP`) o rellenar los espacios con un color (modo de filtro `COLOR`).

Para entender el funcionamiento del filtro de mapa de desplazamiento, se puede considerar el siguiente ejemplo. En el código que aparece a continuación se carga una imagen y, cuando termina la carga, se centra en el escenario y se le aplica un filtro de mapa de desplazamiento que hace que los píxeles de toda la imagen se desplacen horizontalmente hacia la izquierda.

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image3.jpg");
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
    // Center the loaded image on the Stage.
    loader.x = (stage.stageWidth - loader.width) / 2;
    loader.y = (stage.stageHeight - loader.height) / 2;

    // Create the displacement map image.
    mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

    // Create the displacement filter.
    displacementMap = new DisplacementMapFilter();
    displacementMap.mapBitmap = mapImage;
    displacementMap.mapPoint = new Point(0, 0);
    displacementMap.componentX = BitmapDataChannel.RED;
    displacementMap.scaleX = 250;
    loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

Las propiedades utilizadas para definir el desplazamiento son las siguientes:

- Mapa de bits del mapa: el mapa de bits del desplazamiento es una nueva instancia de BitmapData creada por el código. Sus dimensiones coinciden con las de la imagen cargada (de modo que el desplazamiento se aplica a toda la imagen). Tiene un relleno sólido de píxeles rojos.
- Punto del mapa: este valor está ajustado al punto 0, 0, lo que hace que el desplazamiento se aplique a toda la imagen.
- Componente X: este valor está ajustado a la constante `BitmapDataChannel.RED`, lo que significa que el valor rojo del mapa de bits del mapa determinará la cantidad de desplazamiento de los píxeles (cuánto se moverán) en el eje x.
- Escala X: este valor está ajustado a 250. La cantidad total de desplazamiento (teniendo en cuenta que la imagen del mapa es completamente roja) sólo movería la imagen una distancia muy pequeña (aproximadamente medio píxel), de modo que si este valor se ajustase a 1, la imagen sólo se desplazaría 0,5 píxeles horizontalmente. Al ajustarlo a 250, la imagen se mueve aproximadamente 125 píxeles.

Esta configuración hace que los píxeles de la imagen filtrada se desplacen 250 píxeles hacia la izquierda. La dirección (izquierda o derecha) y la cantidad de desplazamiento se basan en el valor de color de los píxeles de la imagen del mapa. Conceptualmente, Flash Player o AIR recorren de uno en uno los píxeles de la imagen filtrada (al menos los píxeles de la región en la que se aplica el filtro, que, en este caso, son todos) y llevan a cabo el siguiente proceso con cada píxel:

1. Buscan el píxel correspondiente en la imagen del mapa. Por ejemplo, cuando Flash Player o AIR calculan la cantidad de desplazamiento para el píxel de la esquina superior izquierda de la imagen filtrada, consultan el píxel de la esquina superior izquierda de la imagen del mapa.
2. Determinan el valor del canal de color especificado en el píxel del mapa. En este caso, el canal de color del componente x es el rojo, de modo que Flash Player y AIR comprueban qué valor tiene el canal rojo de la imagen del mapa en el píxel en cuestión. Dado que la imagen del mapa es de color rojo sólido, el canal rojo del píxel es 0xFF, ó 255. Este será el valor que se use como desplazamiento.
3. Comparan el valor de desplazamiento con el valor "medio" (127, que es el valor intermedio entre 0 y 255). Si el valor de desplazamiento es menor que el valor medio, el píxel se desplaza en dirección positiva (hacia la derecha para el desplazamiento en el eje x y hacia abajo para el desplazamiento en el eje y). Por otro lado, si el valor de desplazamiento es superior al valor medio (como en el ejemplo), el píxel se desplaza en dirección negativa (hacia la izquierda para el desplazamiento en el eje x y hacia arriba para el desplazamiento en el eje y). Para ser más precisos, Flash Player y AIR restan a 127 el valor de desplazamiento y el resultado (positivo o negativo) es la cantidad relativa de desplazamiento que se aplica.
4. Finalmente, determinan la cantidad de desplazamiento real calculando qué porcentaje del desplazamiento completo representa el valor del desplazamiento relativo. En este caso, el rojo sólido significa un desplazamiento del 100%. Ese porcentaje se multiplica por el valor de escala x o escala y a fin de establecer el número de píxeles de desplazamiento que se aplicará. En este ejemplo, la cantidad de desplazamiento equivale a 100% veces un multiplicador de 250 (aproximadamente 125 píxeles a la izquierda).

Dado que no se han especificado valores para el componente y la escala y, se utilizan los valores predeterminados (que no causan ningún desplazamiento), por eso la imagen no se mueve en dirección vertical.

En el ejemplo se utiliza el modo de filtro predeterminado, `WRAP`, de modo que cuando los píxeles se desplazan hacia la izquierda, el espacio vacío de la derecha se llena con los píxeles que se han desplazado fuera del borde izquierdo de la imagen. Se puede experimentar con este valor para ver los distintos efectos. Por ejemplo, si se añade la siguiente línea a la parte del código en la que se definen las propiedades del desplazamiento (antes de la línea `loader.filters = [displacementMap]`), la imagen tendrá la apariencia de haber sido arrastrada por el escenario:

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

A continuación se muestra un ejemplo más complejo, con un listado que utiliza un mapa de desplazamiento para crear un efecto de lupa sobre una imagen:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// Create the gradient circles that will together form the
// displacement map image
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [0xFF0000, 0x000000];
var blueColors:Array = [0x0000FF, 0x000000];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios, xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios, yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// Position the circles at the bottom of the screen, for reference.
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create the map image by combining the two gradient circles.
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false, 0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false, 0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE,
BitmapDataChannel.BLUE);
```

```
yMap.dispose();

// Display the map image on the Stage, for reference.
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// This function creates the displacement map filter at the mouse location.
function magnify():void
{
    // Position the filter.
    var filterX:Number = (loader.mouseX) - (map.width / 2);
    var filterY:Number = (loader.mouseY) - (map.height / 2);
    var pt:Point = new Point(filterX, filterY);
    var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
    xyFilter.mapBitmap = map;
    xyFilter.mapPoint = pt;
    // The red in the map image will control x displacement.
    xyFilter.componentX = BitmapDataChannel.RED;
    // The blue in the map image will control y displacement.
    xyFilter.componentY = BitmapDataChannel.BLUE;
    xyFilter.scaleX = 35;
    xyFilter.scaleY = 35;
    xyFilter.mode = DisplacementMapFilterMode.IGNORE;
    loader.filters = [xyFilter];
}

// This function is called when the mouse moves. If the mouse is
// over the loaded image, it applies the filter.
function moveMagnifier(event:MouseEvent):void
{
    if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
    {
        magnify();
    }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

El código genera primero dos círculos degradados, que se combinan para formar la imagen del mapa de desplazamiento. El círculo rojo crea el desplazamiento en el eje x (`xyFilter.componentX = BitmapDataChannel.RED`), y el azul crea el desplazamiento en el eje y (`xyFilter.componentY = BitmapDataChannel.BLUE`). Para entender mejor la apariencia de la imagen del mapa de desplazamiento, el código añade los círculos originales y el círculo combinado que sirve de imagen del mapa en la parte inferior de la pantalla.

A continuación, el código carga una imagen y, a medida que se mueve el ratón, aplica el filtro de desplazamiento a la parte de la imagen que se encuentra bajo éste. Los círculos degradados que se utilizan como imagen del mapa de desplazamiento hacen que la región desplazada se aleje del puntero del ratón. Hay que tener en cuenta que las regiones grises del mapa de desplazamiento no causan ningún movimiento. El color gris es `0x7F7F7F`. Los canales rojo y azul de esa tonalidad de gris coinciden exactamente con la tonalidad media de esos canales de color, de manera que no hay desplazamiento en las zonas grises de la imagen del mapa. Del mismo modo, en el centro del círculo tampoco hay desplazamiento. Si bien el color de esa región no es gris, los canales rojo y azul de ese color son idénticos a los canales rojo y azul del gris medio y, puesto que el rojo y el azul son los colores que causan el desplazamiento, no se produce ninguno.

Filtro de sombreado

La clase ShaderFilter permite utilizar un efecto de filtro personalizado definido como sombreado de Pixel Bender. Dado que el efecto de filtro se escribe como un sombreado de Pixel Bender, el efecto se puede personalizar completamente. El contenido filtrado se pasa al sombreado en forma de entrada de imagen y el resultado de la operación del sombreado se convierte en el resultado del filtro.

Nota: *el filtro Shader está disponible en ActionScript a partir de Flash Player 10 y Adobe AIR 1.5.*

Para aplicar un filtro de sombreado a un objeto, primero debe crear una instancia de Shader que represente al sombreado de Pixel Bender que se va a utilizar. Para obtener información detallada sobre el procedimiento de creación de una instancia de Shader y el modo de especificar los valores de los parámetros y las imágenes de entrada, consulte "Trabajo con sombreados de Pixel Bender" en la página 395.

Al utilizar un sombreado a modo de filtro, es preciso recordar tres cosas:

- El sombreado se debe definir de forma que acepte al menos una imagen de entrada.

- El objeto filtrado (el objeto de visualización o BitmapData al que se aplica el filtro) se pasa al sombreado como el primer valor de imagen de entrada. Por este motivo, no se debe especificar manualmente un valor para la primera imagen de entrada.
- Si el sombreado define varias imágenes de entrada, las entradas adicionales se deben especificar manualmente (es decir, es preciso establecer la propiedad `input` de cualquier instancia de ShaderInput que pertenezca a la instancia de Shader).

Una vez disponga de un objeto Shader para el sombreado, cree una instancia de ShaderFilter. Este es el objeto de filtro real que se utiliza como cualquier otro filtro. Para crear una clase ShaderFilter que utilice un objeto Shader, llame al constructor de `ShaderFilter()` y pase el objeto Shader en forma de argumento, tal y como se muestra en este listado:

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
```

Para ver un ejemplo completo del uso de un filtro de sombreado, consulte "Utilización de un sombrado como filtro" en la página 413.

Ejemplo: Filter Workbench

Filter Workbench proporciona una interfaz de usuario con la que es posible aplicar distintos filtros a imágenes y otro contenido visual y ver el código resultante que se puede utilizar para generar el mismo efecto en ActionScript. Además de ofrecer una herramienta para la experimentación con filtros, esta aplicación ilustra las siguientes técnicas:

- Creación de instancias de diferentes filtros
- Aplicación de varios filtros a un objeto de visualización

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Filter Workbench se encuentran en la carpeta Samples/FilterWorkbench. La aplicación consta de los siguientes archivos:

Archivo	Descripción
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	Clase que proporciona la principal funcionalidad de la aplicación, incluido el cambio de contenido al que se aplican los filtros y la aplicación de filtros al contenido.
com/example/programmingas3/filterWorkbench/IFilterFactory.as	Interfaz que define los métodos comunes implementados por cada una de las clases Factory de los filtros. Esta interfaz define la funcionalidad común que utiliza la clase FilterWorkbenchController para interactuar con cada clase Factory de los filtros.
En la carpeta com/example/programmingas3/filterWorkbench/: BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	Conjunto de clases, cada una de las cuales implementa la interfaz IFilterFactory. Cada una de estas clases proporciona la funcionalidad de crear y definir valores para un único tipo de filtro. Los paneles de propiedad de filtro de la aplicación utilizan estas clases Factory para crear instancias de sus filtros específicos, los cuales son recuperados y aplicados al contenido de la imagen por la clase FilterWorkbenchController.

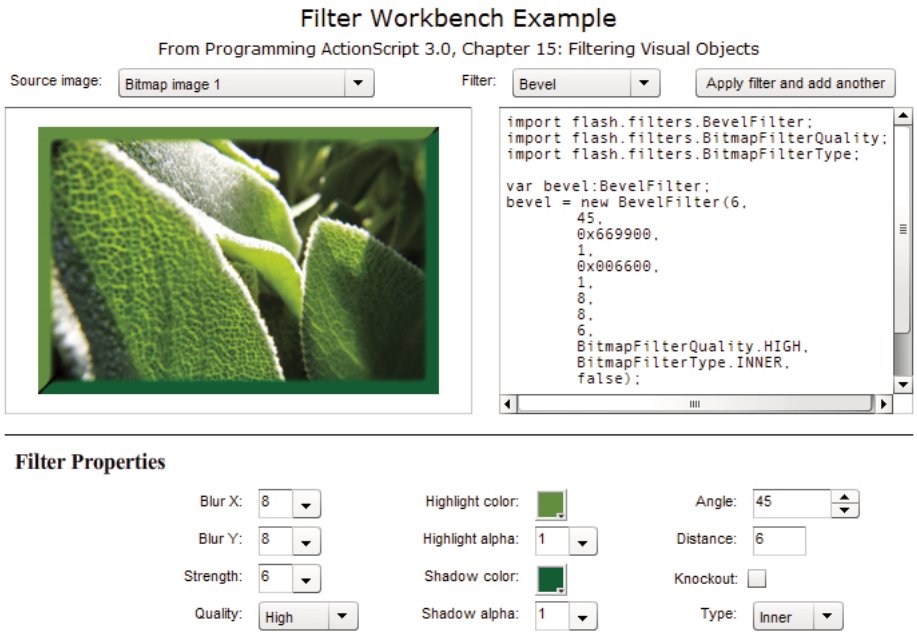
Archivo	**Descripción**
com/example/programmingas3/filterWorkbench/IFilterPanel.as	Interfaz que define los métodos comunes implementados por las clases que, a su vez, definen los paneles de interfaz de usuario utilizados para manipular los valores de filtro en la aplicación.
com/example/programmingas3/filterWorkbench/ColorStringFormatter.as	Clase de utilidad que incluye un método para convertir un valor de color numérico a un formato de cadena hexadecimal.
com/example/programmingas3/filterWorkbench/GradientColor.as	Clase que funciona como un objeto de valor, combinando en un único objeto los tres valores (color, alfa y proporción) asociados a cada color en las clases GradientBevelFilter y GradientGlowFilter.
Interfaz de usuario (Flash)	
FilterWorkbench.fla	El archivo principal que define la interfaz de usuario de la aplicación.
flashapp/FilterWorkbench.as	Clase que proporciona la funcionalidad para la interfaz de usuario principal de la aplicación; esta clase se utiliza como la clase de documento para el archivo FLA de la aplicación.
En la carpeta flashapp/filterPanels: BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	Conjunto de clases que proporciona la funcionalidad para cada panel utilizado para ajustar las opciones para un filtro único. Para cada clase, existe además un símbolo MovieClip asociado en la biblioteca del archivo FLA principal de la aplicación, cuyo nombre coincide con el de la clase (por ejemplo, el símbolo "BlurPanel" está relacionado con la clase definida en BlurPanel.as). Los componentes que conforman la interfaz de usuario se ubican y designan de acuerdo con estos símbolos.
flashapp/ImageContainer.as	Un objeto de visualización que funciona como contenedor para la imagen cargada de la pantalla.
flashapp/BGColorCellRenderer.as	Procesador de celdas personalizado que se utiliza para cambiar el color de fondo de una celda en el componente DataGrid.
flashapp/ButtonCellRenderer.as	Procesador de celdas personalizado que se utiliza para incluir un componente de botón en una celda del componente DataGrid.
Contenido de imagen filtrado	
com/example/programmingas3/filterWorkbench/ImageType.as	Esta clase sirve como objeto de valor que contiene el tipo y el URL del archivo de imagen única en el que la aplicación puede cargar y aplicar los filtros. La clase también incluye un conjunto de constantes que representan los archivos de imagen disponibles.
images/sampleAnimation.swf, images/sampleImage1.jpg, images/sampleImage2.jpg	Imágenes y otro contenido visual al que se aplican los filtros en la aplicación.

Pruebas con filtros de ActionScript

La aplicación Filter Workbench se ha diseñado para ayudar a probar los diferentes efectos de filtro y a generar el código ActionScript necesario para lograr cada uno. La aplicación permite elegir entre tres archivos diferentes que incluyen contenido visual, tales como imágenes de mapa de bits y animaciones creadas por Flash, así como aplicar ocho filtros de ActionScript diferentes a la imagen seleccionada, ya sea de forma individual o en combinación con otros filtros. La aplicación incluye los filtros siguientes:

- Bisel (flash.filters.BevelFilter)
- Desenfoque (flash.filters.BlurFilter)
- Matriz de colores (flash.filters.ColorMatrixFilter)
- Convolución (flash.filters.ConvolutionFilter)
- Sombra (flash.filters.DropShadowFilter)
- Iluminado (flash.filters.GlowFilter)
- Bisel degradado (flash.filters.GradientBevelFilter)
- Iluminado degradado (flash.filters.GradientGlowFilter)

Tras seleccionar una imagen y el filtro que se le va a aplicar, la aplicación muestra un panel con una serie de controles para el ajuste de las propiedades específicas del filtro elegido. Por ejemplo, en la siguiente imagen aparece la aplicación con el filtro de bisel seleccionado:



La vista previa se actualiza en tiempo real conforme se ajustan las propiedades del filtro. También es posible aplicar varios filtros. Para ello, personalice un filtro, haga clic en el botón Aplicar, personalice otro filtro y vuelva a hacer clic en dicho botón, y así sucesivamente.

A continuación se describen algunas funciones y limitaciones de los paneles de filtro de la aplicación:

- El filtro de matriz de colores incluye un grupo de controles que permiten manipular de forma directa las propiedades de imagen comunes, tales como el brillo, el contraste, la saturación y el matiz. Asimismo, es posible especificar valores personalizados de la matriz de colores.
- El filtro de convolución, disponible sólo con ActionScript, incluye un conjunto de valores de matriz de convolución utilizados comúnmente, al tiempo que es posible especificar valores personalizados. No obstante, aunque la clase ConvolutionFilter acepta matrices de cualquier tamaño, la aplicación Filter Workbench utiliza una matriz fija de 3 x 3, el tamaño de filtro que se utiliza con más frecuencia.
- Los filtros de mapa de desplazamiento y de sombreado no se encuentran disponibles en la aplicación Filter Workbench, sólo en ActionScript. El filtro de mapa de desplazamiento requiere una imagen del mapa además del contenido de imagen filtrado. La imagen del mapa del filtro de desplazamiento es la entrada principal que determina el resultado del filtro. Por tanto, si no es posible cargar o crear una imagen del mapa, la capacidad de experimentar con el filtro de desplazamiento de mapa resulta extremadamente limitada. Del mismo modo, el filtro de sombreado requiere un archivo de código de bytes de Pixel Bender además del contenido de imagen filtrado. Si no se carga el código de bytes del sombreado, es imposible utilizar este tipo de filtro.

Creación de instancias de filtros

La aplicación Filter Workbench incluye un conjunto de clases (una para cada filtro disponible) que son utilizadas por los distintos paneles para crear los filtros. Al seleccionar un filtro, el código de ActionScript asociado al panel correspondiente crea una instancia de la clase Factory de filtro apropiada. (Estas clases también se denominan *clases de fábrica* porque su finalidad es crear instancias de otros objetos, del mismo modo que en una fábrica del mundo real se crean productos individuales.)

Cada vez que se cambia el valor de una propiedad en el panel, el código de éste llama al método apropiado de la clase de fábrica. Cada clase de fábrica incluye métodos específicos utilizados por el panel para crear la instancia de filtro apropiada. Por ejemplo, al seleccionar el filtro de desenfoque, la aplicación crea una instancia de BlurFactory. La clase BlurFactory incluye un método `modifyFilter()` que acepta tres parámetros: `blurX`, `blurY`, y `quality`, los cuales se utilizan conjuntamente para crear la instancia de BlurFilter deseada:

```
private var _filter:BlurFilter;

public function modifyFilter(blurX:Number = 4, blurY:Number = 4, quality:int = 1):void
{
    _filter = new BlurFilter(blurX, blurY, quality);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Por otro lado, al seleccionar el filtro de convolución se obtiene una mayor flexibilidad y, en consecuencia, es posible controlar un mayor número de propiedades. En la clase ConvolutionFactory se llama al siguiente código cuando se selecciona un valor diferente en el panel del filtro:

```
private var _filter:ConvolutionFilter;

public function modifyFilter(matrixX:Number = 0,
                                                matrixY:Number = 0,
                                                matrix:Array = null,
                                                divisor:Number = 1.0,
                                                bias:Number = 0.0,
                                                preserveAlpha:Boolean = true,
                                                clamp:Boolean = true,
                                                color:uint = 0,
                                                alpha:Number = 0.0):void
{
    _filter = new ConvolutionFilter(matrixX, matrixY, matrix, divisor, bias, preserveAlpha,
clamp, color, alpha);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Observe en cada caso que, cuando se modifican los valores del filtro, el objeto Factory distribuye un evento `Event.CHANGE` para comunicar a los detectores que los valores del filtro han cambiado. La clase FilterWorkbenchController, responsable de aplicar los filtros al contenido filtrado, detecta dicho evento a fin de determinar cuándo es necesario recuperar una nueva copia del filtro y volver a aplicarla al contenido filtrado.

La clase FilterWorkbenchController no necesita conocer los detalles específicos de la clase de fábrica de cada filtro, sólo saber que el filtro ha cambiado y poder acceder a una copia del mismo. Para que esto sea posible, la aplicación incluye una interfaz, IFilterFactory, que define el comportamiento que debe tener una clase de fábrica de filtro para que la instancia de FilterWorkbenchController de la aplicación pueda cumplir su cometido. IFilterFactory define el método `getFilter`() utilizado en la clase FilterWorkbenchController:

```
function getFilter():BitmapFilter;
```

Observe que la definición del método de interfaz `getFilter()` especifica que devuelve una instancia de BitmapFilter en lugar de un tipo de filtro específico. La clase BitmapFilter no define ningún tipo específico de filtro, sino que se trata de la clase de base a partir de la cual se crean todas las clases de filtro. Cada clase de fábrica de filtro define una implementación específica del método `getFilter()` en la cual se devuelve una referencia al objeto de filtro creado. Por ejemplo, aquí se muestra una versión abreviada del código fuente de la clase ConvolutionFactory:

```
public class ConvolutionFactory extends EventDispatcher implements IFilterFactory
{
    // ------- Private vars -------
    private var _filter:ConvolutionFilter;
    ...
    // ------- IFilterFactory implementation -------
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

En la implementación de la clase ConvolutionFactory del método `getFilter()` se devuelve una instancia de ConvolutionFilter, aunque esto no lo sabrán necesariamente los objetos que llamen a `getFilter()` — según la definición del método `getFilter()` que sigue ConvolutionFactory, debe devolver una instancia de BitmapFilter, que podría ser una instancia de cualquiera de las clases de filtro de ActionScript.

Aplicación de filtros a objetos de visualización

Tal y como se ha explicado en la sección anterior, la aplicación Filter Workbench utiliza una instancia de la clase FilterWorkbenchController (denominada en lo sucesivo "instancia de controlador"), la cual realiza la tarea de aplicar los filtros al objeto visual seleccionado. Antes de que la instancia de controlador pueda aplicar un filtro, debe identificar la imagen o el contenido visual al que se aplicará. Cuando se selecciona una imagen, la aplicación llama al método `setFilterTarget()` de la clase FilterWorkbenchController, pasando una de las constantes definidas en la clase ImageType:

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE, targetLoadComplete);
    ...
}
```

Con esta información, la instancia de controlador carga el archivo designado y, a continuación, lo almacena en una variable de instancia denominada `_currentTarget`:

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

Cuando se selecciona un filtro, la aplicación llama al método `setFilter()` de la instancia de controlador, ofreciendo al controlador una referencia al objeto de fábrica de filtro pertinente, el cual se almacena una variable de instancia denominada `_filterFactory`.

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

Observe que, tal y como se ha descrito anteriormente, la instancia de controlador no conoce el tipo específico de datos de la instancia de fábrica de filtro que se le facilita; sólo sabe que el objeto implementa la instancia de IFilterFactory, lo que significa que incluye un método `getFilter()` y distribuye un evento `change` (`Event.CHANGE`) cuando se modifica el filtro.

Cuando se cambian las propiedades de un filtro en su panel correspondiente, la instancia de controlador detecta que el filtro se ha modificado a través del evento `change` de la fábrica del filtro, evento que llama al método `filterChange()` de la instancia de controlador. Este método, a su vez, llama al método `applyTemporaryFilter()`:

```
private function filterChange(event:Event):void
{
    applyTemporaryFilter();
}

private function applyTemporaryFilter():void
{
    var currentFilter:BitmapFilter = _filterFactory.getFilter();

    // Add the current filter to the set temporarily
    _currentFilters.push(currentFilter);

    // Refresh the filter set of the filter target
    _currentTarget.filters = _currentFilters;

    // Remove the current filter from the set
    // (This doesn't remove it from the filter target, since
    // the target uses a copy of the filters array internally.)
    _currentFilters.pop();
}
```

La tarea de aplicar el filtro al objeto de visualización se produce en el método `applyTemporaryFilter()`. En primer lugar, el controlador recupera una referencia al objeto de filtro llamando al método `getFilter()` de la fábrica de filtro.

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

La instancia de controlador incluye una variable de instancia de Array, denominada `_currentFilters`, que es un conjunto en el cual almacena todos los filtros aplicados al objeto de visualización. El siguiente paso consiste en añadir el filtro recién actualizado a dicho conjunto:

```
_currentFilters.push(currentFilter);
```

A continuación, el código asigna el conjunto de filtros a la propiedad `filters` del objeto de visualización, la cual aplica en realidad los filtros a la imagen:

```
_currentTarget.filters = _currentFilters;
```

Por último, dado que este filtro recién añadido sigue siendo el filtro "de trabajo", conviene no aplicarlo de forma permanente al objeto de visualización, por lo que se elimina del conjunto `_currentFilters`:

```
_currentFilters.pop();
```

La eliminación del filtro no afecta al objeto de visualización filtrado, ya que éste realiza una copia del conjunto de filtros cuando se asigna a la propiedad `filters` y utiliza este conjunto interno en lugar del original. Por este motivo, los cambios realizados en el conjunto de filtros no afectan al objeto de visualización hasta que dicho conjunto se vuelve a asignar a su propiedad `filters`.

Capítulo 17: Trabajo con sombreados de Pixel Bender

El kit de herramientas de Adobe Pixel Bender permite a los desarrolladores escribir sombreados que generan efectos gráficos y llevar a cabo procesamiento adicional de datos e imágenes. El código de bytes de Pixel Bender se puede ejecutar en ActionScript para aplicar el efecto a datos de imagen o contenido visual. El uso de Pixel Bender en ActionScript permite crear efectos visuales personalizados y realizar procesamiento de datos más allá de las capacidades que incorpora ActionScript.

Nota: *la compatibilidad con Pixel Bender existe a partir de Flash Player 10 y Adobe AIR 1.5.*

Fundamentos de los sombreados de Pixel Bender

Introducción a la utilización de los sombreados de Pixel Bender

Adobe Pixel Bender es un lenguaje de programación que se utiliza para crear o manipular contenido de imagen. Con Pixel Bender puede crear un núcleo, también denominado sombreado en este documento. El sombreado define una sola función que se ejecuta en cada uno de los píxeles de una imagen de forma individual. El resultado de cada llamada a la función es el color de salida en las coordenadas del píxel de la imagen. Las imágenes de entrada y los valores de parámetro se pueden especificar para personalizar la operación. En una sola ejecución de un sombreado, los valores de parámetro y entrada son constantes. Lo único que varía son las coordenadas del píxel cuyo color es el resultado de la llamada de la función.

Si es posible, la función de sombreado se llama para varias coordenadas de píxel de salida en paralelo. Esto mejora el rendimiento del sombreado y puede proporcionar un procesamiento de alto rendimiento.

En Flash Player y Adobe AIR, se pueden crear fácilmente tres tipos de efectos utilizando un sombreado:

- Relleno de dibujo
- Modo de mezcla
- Filtro

Un sombreado también se puede ejecutar en modo autónomo. Con el uso del modo autónomo se accede directamente al resultado de un sombreado en lugar de especificar previamente su uso previsto. Se puede acceder al resultado como datos de imagen o datos binarios o numéricos. No es necesario que los datos sean de imagen. De este modo puede proporcionar a un sombreado un conjunto de datos como entrada. El sombreado procesa los datos y puede acceder a los datos de resultado que devuelve.

Nota: *la compatibilidad con Pixel Bender existe a partir de Flash Player 10 y Adobe AIR 1.5.*

Tareas comunes de Pixel Bender

A continuación se enumera una serie de tareas que se suelen llevar a cabo mediante los filtros de ActionScript:

- Carga de un sombreado en una aplicación SWF en ejecución o incorporación del sombreado en tiempo de compilación y acceso al mismo en tiempo de ejecución.
- Acceso a los metadatos de sombreado

- Identificación y especificación de valores para entradas de sombreado (generalmente imágenes).
- Identificación y especificación de valores para parámetros de sombreado.
- Uso de un sombreado de los siguientes modos:
  - Como relleno de dibujo
  - Como modo de mezcla
  - Como filtro
  - En modo autónomo

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Núcleo: para Pixel Bender, un núcleo es lo mismo que un sombreado. Con el uso de Pixel Bender, el código define un núcleo, que define una sola función que se ejecuta en cada uno de los píxeles de una imagen individualmente.
- Código de bytes de Pixel Bender: cuando un núcleo de Pixel Bender se compila, se transforma en un código de bytes de Pixel Bender. Al código de bytes se accede y se ejecuta mediante Flash Player o Adobe AIR en tiempo de ejecución.
- Lenguaje Pixel Bender: lenguaje de programación utilizado para crear un núcleo de Pixel Bender.
- Kit de herramientas de Pixel Bender: la aplicación que se emplea para crear un archivo de código de bytes de Pixel Bender desde el código fuente del lenguaje. El kit de herramientas permite escribir, probar y compilar código fuente de Pixel Bender.
- Sombreado: por motivos de este documento, un sombreado es un conjunto de funcionalidad escrito en el lenguaje Pixel Bender. Un código del sombreado crea un efecto visual o realiza un cálculo. En cualquier caso, el sombreado devuelve un conjunto de datos (generalmente los píxeles de una imagen). El sombreado realiza la misma operación en cada punto de datos, con la única diferencia de las coordenadas del píxel de salida.

  El sombreado no está escrito en ActionScript. Se escribe en el lenguaje Pixel Bender y se compila en el código de bytes de Pixel Bender. Se puede incorporar a un archivo SWF en tiempo de compilación o cargarse como archivo externo en tiempo de ejecución. En cualquier caso se puede acceder al mismo en ActionScript creando un objeto Shader y vinculando este objeto al código de bytes del sombreado.
- Entrada de sombreado: una entrada compleja, generalmente datos de imagen de mapa de bits, que se proporciona a un sombreado para su uso en sus cálculos. Para cada variable de entrada definida en un sombreado, se utiliza un solo valor (es decir, una sola imagen o conjunto de datos binarios) para la ejecución competa del sombreado.
- Parámetro de sombreado: un solo valor (o conjunto limitado de valores) que se proporciona a un sombreado para su uso en sus cálculos. Cada valor de parámetro se define para una sola ejecución del sombreado y el mismo valor se emplea en toda la ejecución.

Ejecución de los ejemplos del capítulo

A medida que progrese en el estudio de este capítulo, podría desear probar los listados de código de ejemplo proporcionados. Como este capítulo se centra en la creación y manipulación de contenido visual, para probar el código hay que ejecutarlo y ver los resultados en el archivo SWF creado. Todos los ejemplos crean contenido utilizando la API de dibujo, que utiliza o se modifica mediante el efecto de sombreado.

La mayor parte de los listados del código de ejemplo incluyen dos partes. Una parte es el código fuente de Pixel Bender para el sombreado utilizado en el ejemplo. En primer lugar debe utilizar el kit de herramientas de Pixel Bender para compilar el código fuente en un archivo de código de bytes de Pixel Bender. Siga estos pasos para crear el archivo de código de bytes de Pixel Bender:

1. Abra el kit de herramientas de Adobe Pixel Bender. Si es necesario, en el menú Build (Compilación), seleccione "Turn on Flash Player warnings and errors" (Activar errores y avisos de Flash Player).
2. Copie el listado de código de Pixel Bender y péguelo en el panel de editor de código de su kit de herramientas.
3. En el menú File (Archivo), seleccione "Export kernel filter for Flash Player" (Exportar filtro de núcleo para Flash Player).
4. Guarde el archivo de código de bytes de Pixel Bender en el mismo directorio que el documento Flash. El nombre del archivo debe coincidir con el nombre especificado en la descripción del ejemplo.

La parte de ActionScript de todos los ejemplos se escribe como un archivo de clase. Para probar la aplicación:

1. Cree un documento de Flash vacío y guárdelo en el equipo.
2. Cree un nuevo archivo de ActionScript y guárdelo en el mismo directorio que el documento de Flash. El nombre del archivo debe coincidir con el nombre de la clase del listado de código. Por ejemplo, si el listado de código define una clase denominada MyApplication, use el nombre MyApplication.as para guardar el archivo de ActionScript.
3. Copie el listado de código en el archivo de ActionScript y guarde el archivo.
4. En el documento de Flash, haga clic en una parte vacía del escenario o espacio de trabajo para activar el inspector de propiedades del documento.
5. En el inspector de propiedades, en el campo Clase de documento, escriba el nombre de la clase de ActionScript que copió del texto.
6. Ejecute el programa utilizando Control > Probar película.

   Verá el resultado del ejemplo en el panel de previsualización.

Estas técnicas para la comprobación de listados de código de ejemplo se explican de forma más detallada en "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

Carga e incorporación de un sombreado

El primer paso para utilizar un sombreado de Pixel Bender en ActionScript es obtener acceso al sombreado en el código ActionScript. Debido a que los sombreados se crean a través del kit de herramientas de Adobe Pixel Bender, y se escriben en el lenguaje de Pixel Bender, no se puede acceder directamente a ellos en ActionScript. En su lugar, se crea una instancia de la clase Shader que representa el sombreado de Pixel Bender en ActionScript. El objeto Shader permite encontrar información sobre el sombreado. Así, permite determinar si el sombreado espera parámetros o valores de imagen de entrada. El objeto Shader se pasa a otros objetos para utilizar el sombreado. Por ejemplo, para utilizar el sombreado como un filtro, el objeto Shader se asigna a la propiedad `shader` de un objeto ShaderFilter. Como alternativa, para utilizar el sombreado como un relleno de dibujo, el objeto Shader se pasa como un argumento al método `Graphics.beginShaderFill()`.

El código ActionScript puede acceder a un sombreado creado por el kit de herramientas de Adobe Pixel Bender (un archivo .pbj) de dos formas diferentes:

- Cargado en tiempo de ejecución: el archivo de sombreado se puede cargar como un activo externo a través de un objeto URLLoader. Esta técnica es similar a cargar un activo externo, como un archivo de texto. En el siguiente ejemplo se muestra la carga de un archivo de código de bytes de sombreado en tiempo de ejecución y su vinculación a una instancia de Shader:

```
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.BINARY;
loader.addEventListener(Event.COMPLETE, onLoadComplete);
loader.load(new URLRequest("myShader.pbj"));

var shader:Shader;

function onLoadComplete(event:Event):void {
    // Create a new shader and set the loaded data as its bytecode
    shader = new Shader();
    shader.byteCode = loader.data;

    // You can also pass the bytecode to the Shader() constructor like this:
    // shader = new Shader(loader.data);

     // do something with the shader
}
```

- Incorporado en el archivo SWF: el archivo de sombreado se puede incorporar en el archivo SWF en tiempo de compilación a través de la etiqueta de metadatos `[Embed]`. La etiqueta de metadatos `[Embed]` sólo está disponible si utiliza el SDK de Flex para compilar el archivo SWF. El parámetro `[Embed]` de la etiqueta `source` apunta al archivo de sombreado, y su parámetro `mimeType` es `"application/octet-stream"`, como se muestra en este ejemplo:

```
[Embed(source="myShader.pbj", mimeType="application/octet-stream")]
var MyShaderClass:Class;

// ...

// create a shader and set the embedded shader as its bytecode
var shaderShader = new Shader();
shader.byteCode = new MyShaderClass();

// You can also pass the bytecode to the Shader() constructor like this:
// var shader:Shader = new Shader(new MyShaderClass());

// do something with the shader
```

En cualquier caso, se vincula el código de bytes de sombreado sin procesar (la propiedad `URLLoader.data` o una instancia de la clase de datos `[Embed]`) a la instancia de Shader. Como se muestra en el ejemplo anterior, el código de bytes se puede asignar a la instancia de Shader de dos modos diferentes. Puede transferir el código de bytes de sombreado como un argumento al constructor `Shader()`. También puede establecerlo como la propiedad `byteCode` de la instancia de Shader.

Una vez que se ha creado un sombreado de Pixel Bender y se ha vinculado a un objeto Shader, el sombreado se puede utilizar para crear efectos de formas diferentes. Se puede utilizar como filtro, modo de mezcla, relleno de mapa de bits o para la reproducción autónoma de mapas de bits u otros datos. La propiedad `data` del objeto Shader se puede utilizar para acceder a los metadatos del sombreado, especificar imágenes de entrada y establecer valores de parámetro.

Acceso a los metadatos de sombreado

Al crear un núcleo de sombreado de Pixel Bender, el autor puede especificar los metadatos del sombreado en el código fuente de Pixel Bender. Al utilizar un sombreado en ActionScript, puede examinar el sombreado y extraer sus metadatos.

Al crear una instancia de Shader y vincularla a un sombreado de Pixel Bender, se crea un objeto ShaderData que contiene datos sobre el sombreado y se almacena en la propiedad `data` del objeto Shader. La clase ShaderData no define propiedades propias. Sin embargo, en tiempo de ejecución una propiedad se añade dinámicamente al objeto ShaderData para cada valor de metadatos definido en el código fuente del sombreado. El nombre dado a cada propiedad es el mismo que el especificado en los metadatos. Por ejemplo, supongamos que el código fuente de un sombreado de Pixel Bender incluye la siguiente definición de metadatos:

```
namespace : "Adobe::Example";
vendor : "Bob Jones";
version : 1;
description : "Creates a version of the specified image with the specified brightness.";
```

El objeto ShaderData creado para dicho sombreado se genera con las propiedades y valores siguientes:

- `namespace` (String): `"Adobe::Example"`
- `vendor` (String): `"Bob Jones"`
- `version` (String): `"1"`
- `description` (String): `"Crea una versión de la imagen especificada con el brillo especificado"`

Debido a que las propiedades de metadatos se añaden dinámicamente al objeto ShaderData, el bucle `for..in` se puede utilizar para examinar el objeto ShaderData. Esta técnica permite determinar si el sombreado contiene metadatos y cuáles son los valores de éstos. Además de las propiedades de metadatos, los objetos ShaderData pueden contener propiedades que representan entradas y parámetros definidos en el sombreado. Al utilizar un bucle `for..in` para examinar un objeto ShaderData, es necesario comprobar los tipos de datos de cada propiedad para determinar si la propiedad es una entrada (una instancia de ShaderInput), un parámetro (una instancia de ShaderParameter) o un valor de metadatos (una instancia de String). El ejemplo siguiente muestra cómo utilizar un bucle `for..in` para examinar las propiedades dinámicas de la propiedad `data` de un sombreado. Cada valor de metadatos se añade a una instancia de Vector denominada `metadata`. Es necesario tener en cuenta que en este ejemplo se asume que ya se ha creado una instancia de Shader denominada `myShader`:

```
var shaderData:ShaderData = myShader.data;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (!(shaderData[prop] is ShaderInput) && !(shaderData[prop] is ShaderParameter))
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the metadata
```

Para obtener una versión de este ejemplo que también extrae entradas y parámetros de sombreado, consulte "Identificación de entradas y parámetros de sombreado" en la página 400. Para obtener más información sobre las propiedades de entrada y parámetro, consulte "Especificación de valores de entrada y parámetro de sombreado" en la página 400.

Especificación de valores de entrada y parámetro de sombreado

Un gran número de sombreados de Pixel Bender se definen para utilizar una o varias imágenes de entrada que se utilizan en el procesamiento de sombreado. Por ejemplo, es habitual que un sombreado acepte una imagen fuente y dar lugar a dicha imagen con un efecto determinado aplicado en la misma. En función de cómo se utilice el sombreado, el valor de entrada se podrá especificar automáticamente o quizá será necesario proporcionar un valor de forma explícita. Asimismo, numerosos sombreados especifican parámetros que se utilizan para personalizar la salida del sombreado. Para poder utilizar el sombreado, también debe establecer explícitamente un valor para cada parámetro.

La propiedad `data` del objeto Shader se utiliza para establecer entradas y parámetros de sombreado y para determinar si un sombreado específico espera entradas o parámetros. La propiedad `data` es una instancia de ShaderData.

Identificación de entradas y parámetros de sombreado

El primer paso para especificar los valores de entrada y parámetro de sombreado es determinar si el sombreado que se utiliza espera imágenes o parámetros de entrada. Cada instancia de Shader tiene una propiedad `data` que contiene un objeto ShaderData. Si el sombreado define entradas o parámetros, a éstos se accede como propiedades de dicho objeto ShaderData. Los nombres de las propiedades coinciden con los nombres especificados para las entradas y los parámetros del código fuente del sombreado. Por ejemplo, si un sombreado define una entrada `src`, el objeto ShaderData tiene una propiedad `src` que representa dicha entrada. Cada propiedad que representa una entrada es una instancia de ShaderInput, y cada propiedad que representa un parámetro es una instancia de ShaderParameter.

Lo ideal es que el autor del sombreado proporcione la documentación para dicho sombreado, indicando los valores de imagen de entrada y los parámetros que espera el sombrado, lo que representan, los valores adecuados, etc.

Sin embargo, si el sombreado no está documentado (y no se dispone de su código fuente), se pueden examinar los datos del sombreado para identificar las entradas y los parámetros. Las propiedades que representan las entradas y los parámetros se añaden de forma dinámica al objeto ShaderData. En consecuencia, se puede utilizar un bucle `for..in` para examinar el objeto ShaderData y determinar si su sombreado asociado define entradas o parámetros. Como se describe en "Acceso a los metadatos de sombreado" en la página 399, a los valores de metadatos definidos para un sombrado también se accede como una propiedad dinámica añadida a la propiedad `Shader.data`. Al utilizar esta técnica para identificar las entradas y los parámetros de sombreado, compruebe los tipos de datos de las propiedades dinámicas. Si una propiedad es una instancia de ShaderInput dicha propiedad representa una entrada. Si es una instancia de ShaderParameter, representa un parámetro. De lo contrario, es un valor de metadatos. El ejemplo siguiente muestra cómo utilizar un bucle `for..in` para examinar las propiedades dinámicas de la propiedad `data` de un sombreado. Cada entrada (objeto ShaderInput) se añade a la instancia de Vector `inputs`. Cada parámetro (objeto ShaderParameter) se añade a la instancia de Vector `parameters`. Por último, las propiedades de metadatos se añaden a la instancia de Vector `metadata`. Es preciso tener en cuenta que en este ejemplo se asume que ya se ha creado la instancia de Shader `myShader`:

```
var shaderData:ShaderData = myShader.data;
var inputs:Vector.<ShaderInput> = new Vector.<ShaderInput>();
var parameters:Vector.<ShaderParameter> = new Vector.<ShaderParameter>();
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (shaderData[prop] is ShaderInput)
    {
        inputs[inputs.length] = shaderData[prop];
    }
    else if (shaderData[prop] is ShaderParameter)
    {
        parameters[parameters.length] = shaderData[prop];
    }
    else
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the inputs or properties
```

Especificación de valores de entrada de sombreado

Un gran número de sombreados esperan una o varias imágenes de entrada que se utilizan en el procesamiento de sombreados. Sin embargo, en un gran número de casos se especifica una entrada automáticamente al utilizar el objeto Shader. Por ejemplo, si un sombreado requiere una entrada, y dicho sombreado se utiliza como filtro. Cuando el filtro se aplica a un objeto de visualización o un objeto BitmapData, dicho objeto se establece automáticamente con la entrada. En este caso no se establece explícitamente un valor de entrada.

No obstante, en algunos casos, en especial si un sombreado define varias entradas, el valor para la entrada se establece explícitamente. Cada entrada que se define en un sombreado viene representada en ActionScript por un objeto ShaderInput. El objeto ShaderInput es una propiedad de la instancia de ShaderData de la propiedad `data` del objeto Shader, como se describe en "Identificación de entradas y parámetros de sombreado" en la página 400. Por ejemplo, si un sombreado define una entrada denominada `src` y dicho sombreado está vinculado a un objeto Shader denominado `myShader`. En este caso, se accede al objeto ShaderInput correspondiente a la entrada `src` a través del identificador siguiente:

```
myShader.data.src
```

Cada objeto ShaderInput tiene una propiedad `input` que se utiliza para establecer el valor de la entrada. La propiedad `input` se establece como una instancia de BitmapData para especificar datos de imagen. La propiedad `input` también se puede establecer como BitmapData o Vector.<Number> instancia para especificar datos binarios o numéricos. Para obtener información detallada y conocer las restricciones relativas a la utilización de BitmapData o Vector.<Number> instancia como entrada, consulte el listado de `ShaderInput.input` en la referencia del lenguaje.

Además de la propiedad `input`, el objeto ShaderInput presenta propiedades que se pueden utilizar para determinar el tipo de imagen que espera la entrada. Entre estas propiedades se incluyen `width`, `height` y `channels`. Cada objeto ShaderInput también tiene una propiedad `index` que resulta útil para determinar si se debe proporcionar un valor explícito para la entrada. Si un sombreado espera más entradas de las establecidas automáticamente, se deberán establecer valores para dichas entradas. Para obtener información detallada sobre las diferentes formas de utilizar un sombreado, y si los valores de entrada se establecen automáticamente, consulte "Utilización de un sombreado" en la página 405.

Especificación de valores de parámetro de sombreado

Algunos sombreados definen valores de parámetro que el sombreado utiliza al crear su resultado. Por ejemplo, un sombreado que modifica el brillo de una imagen podría especificar un parámetro de brillo que determine el grado en que la operación afecta el brillo. Un solo parámetro definido en un sombreado puede esperar uno o varios valores, en función de la definición de parámetro en el sombreado. Cada parámetro que se define en un sombreado está representado en ActionScript por un objeto ShaderParameter. El objeto ShaderParameter es una propiedad de la instancia de ShaderData de la propiedad data del objeto Shader, como se describe en "Identificación de entradas y parámetros de sombreado" en la página 400. Por ejemplo, si un sombreado define un parámetro denominado `brightness`, y dicho sombreado está representado por un objeto Shader denominado `myShader`. En este caso se accede al objeto ShaderParameter correspondiente al parámetro `brightness` a través del identificador siguiente:

```
myShader.data.brightness
```

Para establecer un valor (o varios valores) para el parámetro, cree un conjunto ActionScript que contenga el valor o los valores y asígnela a la propiedad `value` del objeto ShaderParameter. La propiedad `value` se define como una instancia de Array porque es posible que un solo parámetro de sombreado requiera varios valores. Incluso si el parámetro de sombreado espera únicamente un solo valor, este valor se debe incluir en un objeto Array para asignarlo a la propiedad `ShaderParameter.value`. El listado siguiente muestra cómo establecer un solo valor con la propiedad `value`:

```
myShader.data.brightness.value = [75];
```

Si el código fuente de Pixel Bender del sombreado define un valor predeterminado para el parámetro, se creará un conjunto que contendrá el valor o los valores predeterminados y se asignará a la propiedad `value` del ShaderParameter al crear el objeto Shader. Una vez que se ha asignado un conjunto a la propiedad `value` (incluso si se trata del conjunto predeterminado), se podrá modificar el valor de parámetro cambiando el valor del elemento de conjunto. No es necesario crear un conjunto nuevo y asignarlo a la propiedad `value`.

En el ejemplo siguiente se muestra cómo establecer un valor de parámetro de sombreado en ActionScript. En este ejemplo el sombrado define el parámetro `color`. El parámetro `color` se declara como una variable `float4` en el código fuente de Pixel Bender, por lo que se trata de un conjunto de cuatro números de coma flotante. En el ejemplo, el valor del parámetro `color` cambia continuamente, y cada vez que cambia el sombreado se utiliza para dibujar un rectángulo de color en la pantalla. El resultado es un cambio de color animado.

Nota: *el código de este ejemplo fue escrito por Ryan Taylor. Gracias Ryan por compartir este ejemplo. Para obtener información sobre el trabajo de Ryan, consulte www.boostworthy.com/.*

El código ActionScript se centra en tres métodos:

- `init()`: en el método `init()` el código carga el archivo de código de bytes de Pixel Bender que contiene el sombreado. Cuando se carga el archivo, se llama al método `onLoadComplete()`.
- `onLoadComplete()`: en el método `onLoadComplete()` el código crea el objeto Shader denominado `shader`. También se crea una instancia de Sprite denominada `texture`. En el método `renderShader()`, el código dibuja el resultado del sombreado en `texture` una sola vez por fotograma.
- `onEnterFrame()`: el método `onEnterFrame()` se llama una vez en cada fotograma, lo que crea el efecto de animación. En este método, el código establece el valor del parámetro del sombreado en el color nuevo y, a continuación, llama al método `renderShader()` para dibujar el resultado del sombreado como un rectángulo.
- `renderShader()`: en el método `renderShader()`, el código llama al método `Graphics.beginShaderFill()` para especificar un relleno de sombreado. A continuación, dibuja un rectángulo cuyo relleno viene definido por la salida del sombreado (el color generado). Para obtener más información sobre la utilización de un sombreado de este modo, consulte "Utilización de un sombreado como relleno de dibujo" en la página 405.

A continuación se incluye el código de ActionScript de este ejemplo. Utilice esta clase como clase principal de aplicación para un proyecto sólo ActionScript en Flex, o bien, como la clase de documento para el archivo FLA en la herramienta de edición de Flash:

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ColorFilterExample extends Sprite
    {
        private const DELTA_OFFSET:Number = Math.PI * 0.5;
        private var loader:URLLoader;
        private var shader:Shader;
        private var texture:Sprite;
        private var delta:Number = 0;

        public function ColorFilterExample()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ColorFilter.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            shader.data.point1.value = [topMiddle.x, topMiddle,y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];
            texture = new Sprite();

            addChild(texture);

            addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void
        {
```

```
            shader.data.color.value[0] = 0.5 + Math.cos(delta - DELTA_OFFSET) * 0.5;
            shader.data.color.value[1] = 0.5 + Math.cos(delta) * 0.5;
            shader.data.color.value[2] = 0.5 + Math.cos(delta + DELTA_OFFSET) * 0.5;
            // The alpha channel value (index 3) is set to 1 by the kernel's default
            // value. This value doesn't need to change.

            delta += 0.1;

            renderShader();
        }

        private function renderShader():void
        {
            texture:graphics.clear();
            texture.graphics.beginShaderFill(shader);
            texture.graphics.drawRect(0, 0, stage.stageWidth, stage.stageHeight);
            texture.graphics.endFill();
        }
    }
}
```

A continuación se incluye el código fuente del núcleo del sombreado ColorFilter, que se utiliza para crear el archivo de código de bytes "ColorFilter.pbj" de Pixel Bender:

```
<languageVersion : 1.0;>
kernel ColorFilter
<
    namespace : "boostworthy::Example";
    vendor : "Ryan Taylor";
    version : 1;
    description : "Creates an image where every pixel has the specified color value.";
>
{
    output pixel4 result;

    parameter float4 color
    <
        minValue:float4(0, 0, 0, 0);
        maxValue:float4(1, 1, 1, 1);
        defaultValue:float4(0, 0, 0, 1);
    >;

    void evaluatePixel()
    {
        result = color;
    }
}
```

Si utiliza un sombreado cuyos parámetros no están documentados, podrá determinar cuántos elementos y de qué tipo se deben incluir en el conjunto si comprueba la propiedad `type` del objeto ShaderParameter. La propiedad `type` indica el tipo de datos del parámetro como se define en el propio sombreado. Para obtener una lista del número y tipo de elementos que espera cada tipo de parámetro, consulte el listado de la propiedad `ShaderParameter.value` en la referencia del lenguaje.

Cada objeto ShaderParameter también tiene una propiedad `index` que indica la ubicación adecuada del parámetro en el orden de los parámetros del sombreado. Además de estas propiedades, los objetos ShaderParameter pueden presentar propiedades adicionales que contienen valores de metadatos proporcionados por el autor del sombreado. Por ejemplo, el autor puede especificar los valores de metadatos mínimo, máximo y predeterminado para un parámetro determinado. Los valores de metadatos que el autor especifica se añaden al objeto ShaderParameter como propiedades dinámicas. Para examinar estas propiedades, utilice el bucle `for..in` para recorrer las propiedades dinámicas del objeto ShaderParameter e identificar sus metadatos. El ejemplo siguiente muestra cómo utilizar un bucle `for..in` para identificar los metadatos de un objeto ShaderParameter. Cada valor de metadatos se añade a una instancia de Vector denominada `metadata`. Tenga en cuenta que este ejemplo asume que las instancias de Shader `myShader` ya están creadas, y que tienen un parámetro `brightness`:

```
var brightness:ShaderParameter = myShader.data.brightness;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in brightness)
{
    if (brightness[prop] is String)
    {
        metadata[metadata.length] = brightness[prop];
    }
}

// do something with the metadata
```

Utilización de un sombreado

Una vez que un sombreado de Pixel Bender está disponible en ActionScript como un objeto Shader, se puede utilizar de varias formas:

- Relleno de dibujo de sombreado: el sombreado define la parte de relleno de una forma dibujada con la API de dibujo
- Modo de mezcla: el sombreado define la mezcla entre dos objetos de visualización solapadas
- Filtro: el sombreado define un filtro que modifica el aspecto del contenido visual
- Procesamiento autónomo de sombreados: el procesamiento de sombreados se ejecuta sin especificar el uso pretendido de la salida. El sombreado se puede ejecutar opcionalmente en el fondo, y estará disponible cuando el procesamiento se complete. Esta técnica se puede utilizar para generar datos de mapa de bits y procesar datos no visuales.

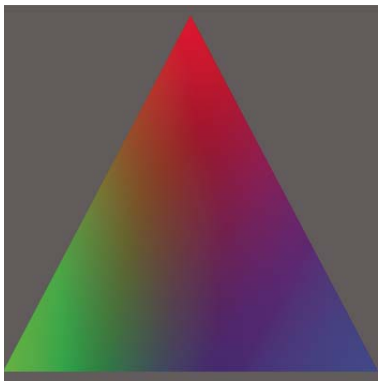
Utilización de un sombreado como relleno de dibujo

Cuando se utiliza un sombreado para crear un relleno de dibujo, se emplean los métodos de la API de dibujo para crear una forma vectorial. La salida del sombreado se usa para rellenar la forma, del mismo modo que cualquier imagen de mapa de bits se puede utilizar como relleno de mapa de bits con la API de dibujo. Para crear un relleno de sombreado, en el punto del código en el que desea comenzar a dibujar la forma, llame al método `beginShaderFill()` del objeto Graphics. Pase el objeto Shader como primer argumento al método `beginShaderFill()`, tal y como se muestra en este listado:

```
var canvas:Sprite = new Sprite();
canvas.graphics.beginShaderFill(myShader);
canvas.graphics.drawRect(10, 10, 150, 150);
canvas.graphics.endFill();
// add canvas to the display list to see the result
```

Cuando se utiliza un sombreado como relleno de dibujo, se establecen todos los valores de la imagen de entrada y los valores de parámetro que requiere el sombreado.

En el siguiente ejemplo se muestra el uso de un sombreado como relleno de dibujo. En este ejemplo, el sombreado crea un degradado de tres puntos. Este degradado tiene tres colores, cada uno de ellos en cada punto de un triángulo, con una mezcla de degradado entre los mismos. Asimismo, los colores giran para crear un efecto de color giratorio animado.



Nota: *Petri Leskinen ha escrito el código de este ejemplo. Gracias Petri por compartir este ejemplo. Para ver más ejemplos y tutoriales de Petri, consulte http://pixelero.wordpress.com/.*

El código de ActionScript se encuentra en tres métodos:

- `init()`: el método `init()` se llama cuando se carga la aplicación. En este método, el código establece los valores iniciales para los objetos Point que representan los puntos del triángulo. El código también crea una instancia de Sprite denominada `canvas`. Posteriormente, en `updateShaderFill()`, el código dibuja el resultado del sombreado en `canvas` una vez por fotograma. Finalmente, el código carga el archivo de código de bytes del sombreado.
- `onLoadComplete()`: en el método `onLoadComplete()` el código crea el objeto Shader denominado `shader`. También establece los valores de parámetro iniciales. Finalmente, el código añade el método `updateShaderFill()` como detector para el evento `enterFrame`, lo que significa que se llama una vez por fotograma para crear un efecto de animación.
- `updateShaderFill()`: el método `updateShaderFill()` se llama una vez en cada fotograma, lo que crea el efecto de animación. En este método, el código calcula y define los valores de los parámetros de sombreado. A continuación el código llama al método `beginShaderFill()` para crear un relleno de sombreado y llama a otros métodos de la API de dibujo para dibujar el resultado de sombreado en un triángulo.

A continuación se incluye el código de ActionScript de este ejemplo. Utilice esta clase como clase principal de aplicación para un proyecto sólo ActionScript en Flex, o bien, como la clase de documento para el archivo FLA en la herramienta de edición de Flash:

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Point;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ThreePointGradient extends Sprite
    {
        private var canvas:Sprite;
        private var shader:Shader;
        private var loader:URLLoader;

        private var topMiddle:Point;
        private var bottomLeft:Point;
        private var bottomRight:Point;

        private var colorAngle:Number = 0.0;
        private const d120:Number = 120 / 180 * Math.PI; // 120 degrees in radians


        public function ThreePointGradient()
        {
            init();
        }

        private function init():void
        {
            canvas = new Sprite();
            addChild(canvas);

            var size:int = 400;
            topMiddle = new Point(size / 2, 10);
            bottomLeft = new Point(0, size - 10);
            bottomRight = new Point(size, size - 10);

            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ThreePointGradient.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            shader.data.point1.value = [topMiddle.x, topMiddle,y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];

            addEventListener.Event.ENTER_FRAME, updateShaderFill);
        }

        private function updateShaderFill(event:Event):void
```

```
        {
            colorAngle += .06;

            var c1:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle);
            var c2:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle + d120);
            var c3:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle - d120;

            shader.data.color1.value = [c1, c2, c3, 1.0];
            shader.data.color2.value = [c3, c1, c2, 1.0];
            shader.data.color3.value = [c2, c3, c1, 1.0];

            canvas.graphics.clear();
            canvas.graphics.beginShaderFill(shader);

            canvas.graphics.moveTo(topMiddle.x, topMiddle.y);
            canvas.graphics.lineTo(bottomLeft.x, bottomLeft.y);
            canvas.graphics.lineTo(bottomRight.x, bottomLeft.y);

            canvas.graphics.endFill();
        }
    }
}
```

Este es el código fuente del núcleo del sombreado ThreePointGradient, que se utiliza para crear el archivo de código de bytes "ThreePointGradient.pbj" de Pixel Bender:

```
<languageVersion : 1.0;>
kernel ThreePointGradient
<
    namespace : "Petri Leskinen::Example";
    vendor : "Petri Leskinen";
    version : 1;
    description : "Creates a gradient fill using three specified points and colors.";
>
{
    parameter float2 point1 // coordinates of the first point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 0);
    >;

    parameter float4 color1 // color at the first point, opaque red by default
    <
        defaultValue:float4(1.0, 0.0, 0.0, 1.0);
    >;

    parameter float2 point2 // coordinates of the second point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color2 // color at the second point, opaque green by default
    <
        defaultValue:float4(0.0, 1.0, 0.0, 1.0);
```

```
    >;

    parameter float2 point3 // coordinates of the third point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color3 // color at the third point, opaque blue by default
    <
        defaultValue:float4(0.0, 0.0, 1.0, 1.0);
    >;

    output pixel4 dst;

    void evaluatePixel()
    {
        float2 d2 = point2 - point1;
        float2 d3 = point3 - point1;

        // transformation to a new coordinate system
        // transforms point 1 to origin, point2 to (1, 0), and point3 to (0, 1)
        float2x2 mtrx = float2x2(d3.y, -d2.y, -d3.x, d2.x) / (d2.x * d3.y - d3.x * d2.y);
        float2 pNew = mtrx * (outCoord() - point1);

        // repeat the edge colors on the outside
        pNew.xy = clamp(pNew.xy, 0.0, 1.0); // set the range to 0.0 ... 1.0

        // interpolating the output color or alpha value
        dst = mix(mix(color1, color2, pNew.x), color3, pNew.y);
    }
}
```

Para obtener más información sobre el dibujo de formas utilizando la API de dibujo, consulte "Utilización de la API de dibujo" en la página 329.

Utilización de un sombreado como modo de mezcla

La utilización de un sombreado como modo de mezcla es similar al uso de otros modos de mezcla. El sombreado define la apariencia que se obtiene de dos objetos de visualización que se mezclan de forma conjunta visualmente. Para utilizar un sombreado como modo de mezcla, asigne el objeto Shader a la propiedad `blendShader` del objeto de visualización del primer plano. Si se asigna una valor distinto a `null` a la propiedad `blendShader`, la propiedad `blendMode` del objeto de visualización se establece automáticamente en `BlendMode.SHADER`. En el siguiente listado se muestra el uso de un sombreado como modo de mezcla. Observe que en este ejemplo se da por hecho que existe un objeto de visualización llamado `foreground`, que se incluye en el mismo elemento principal de la lista de visualización que el resto de contenido de visualización, con el valor `foreground` que se superpone al resto de contenido:

```
foreground.blendShader = myShader;
```

Cuando se utiliza un sombreado como modo de mezcla, el sombreado se debe definir con dos entradas como mínimo. Tal y como muestra el ejemplo, en el código no se definen los valores de entrada. Sin embargo, las dos imágenes mezcladas se utilizan automáticamente como entradas del sombreado. La imagen en primer plano se establece como segunda imagen. (Este es el objeto de visualización al que se aplica el modo de mezcla.) Una imagen de primer plano se crea adoptando un compuesto de todos los píxeles detrás del cuadro delimitador de la imagen de primer plano. Esta imagen se establece como la primera imagen de entrada. Si se utiliza un sombreado que espera más de dos entradas, se proporciona un valor para todas las entradas posteriores a las dos primeras.

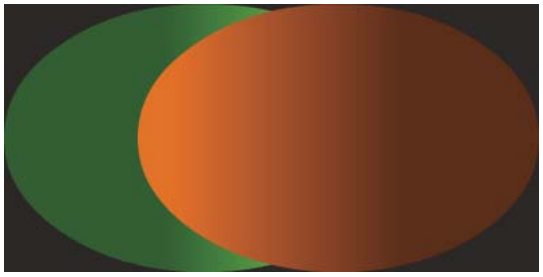
En el siguiente ejemplo se muestra el uso de un sombreado como modo de mezcla. El ejemplo utiliza un modo de mezcla de aclarado basado en la luminosidad. El resultado de la mezcla es que el valor de píxel más claro de cualquiera de los objetos mezclados pasa a ser el píxel que se muestra.

Nota: *Mario Klingemann ha escrito el código de este ejemplo. Gracias Mario por compartir este ejemplo. Para obtener más información sobre el trabajo de Mario y consultar su material, visite www.quasimondo.com/.*

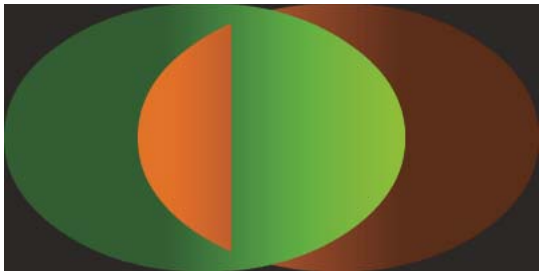
El código de ActionScript importante se encuentra en estos dos métodos:

- `init()`: el método `init()` se llama cuando se carga la aplicación. En este método el código carga el archivo de código de bytes de sombreado.
- `onLoadComplete()`: en el método `onLoadComplete()` el código crea el objeto Shader denominado `shader`. A continuación dibuja tres objetos. El primero, `backdrop`, es un fondo gris oscuro detrás de los objetos mezclados. El segundo, `backgroundShape`, es una elipse con degradado verde. El tercer objeto, `foregroundShape`, es una elipse con degradado naranja.

  La elipse `foregroundShape` es el objeto de primer plano de la mezcla. La imagen de fondo de la mezcla está formada por la parte de `backdrop` y la parte de `backgroundShape`, que se superponen mediante al cuadro delimitador del objeto `foregroundShape`. El objeto `foregroundShape` es el objeto de primer orden en la lista de visualización. Se superpone parcialmente a `backgroundShape` y completamente a `backdrop`. Debido a este solapamiento, si no se aplica un modo de mezcla, la elipse naranja (`foregroundShape`) se muestra completamente y parte de la elipse verde (`backgroundShape`) queda oculta:



  Sin embargo, cuando se aplica el modo de mezcla, la parte más clara de la elipse verde se “deja ver”, ya que es más clara que la parte de `foregroundShape` que se solapa con ella:



Este es el código de ActionScript de este ejemplo. Utilice esta clase como clase principal de aplicación para un proyecto sólo ActionScript en Flex, o bien, como la clase de documento para el archivo FLA en la herramienta de edición de Flash:

```
package
{
    import flash.display.BlendMode;
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class LumaLighten extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function LumaLighten()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("LumaLighten.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var backdrop:Shape = new Shape();
            var g0:Graphics = backdrop.graphics;
            g0.beginFill(0x303030);
            g0.drawRect(0, 0, 400, 200);
            g0.endFill();
            addChild(backdrop);

            var backgroundShape:Shape = new Shape();
            var g1:Graphics = backgroundShape.graphics;
            var c1:Array = [0x336600, 0x80ff00];
            var a1:Array = [255, 255];
            var r1:Array = [100, 255];
            var m1:Matrix = new Matrix();
            m1.createGradientBox(300, 200);
            g1.beginGradientFill(GradientType.LINEAR, c1, a1, r1, m1);
            g1.drawEllipse(0, 0, 300, 200);
```

```
            g1.endFill();
            addChild(backgroundShape);

            var foregroundShape:Shape = new Shape();
            var g2:Graphics = foregroundShape.graphics;
            var c2:Array = [0xff8000, 0x663300];
            var a2:Array = [255, 255];
            var r2:Array = [100, 255];
            var m2:Matrix = new Matrix();
            m2.createGradientBox(300, 200);
            g2.beginGradientFill(GradientType.LINEAR, c2, a2, r2, m2);
            g2.drawEllipse(100, 0, 300, 200);
            g2.endFill();
            addChild(foregroundShape);

            foregroundShape.blendShader = shader;
            foregroundShape.blendMode = BlendMode.SHADER;
        }
    }
}
```

Este es el código fuente del núcleo del sombreado LumaLighten, que se utiliza para crear el archivo de código de bytes "LumaLighten.pbj" de Pixel Bender:

```
<languageVersion : 1.0;>
kernel LumaLighten
<
    namespace : "com.quasimondo.blendModes";
    vendor : "Quasimondo.com";
    version : 1;
    description : "Luminance based lighten blend mode";
>
{
    input image4 background;
    input image4 foreground;

    output pixel4 dst;

    const float3 LUMA = float3(0.212671, 0.715160, 0.072169);

    void evaluatePixel()
    {
        float4 a = sampleNearest(foreground, outCoord());
        float4 b = sampleNearest(background, outCoord());
        float luma_a = a.r * LUMA.r + a.g * LUMA.g + a.b * LUMA.b;
        float luma_b = b.r * LUMA.r + b.g * LUMA.g + b.b * LUMA.b;

        dst = luma_a > luma_b ? a : b;
    }
}
```

Para obtener más información sobre el uso de los modos de mezcla, consulte "Aplicación de modos de mezcla" en la página 312.

Utilización de un sombrado como filtro

La utilización de un sombreado como filtro es similar al uso de cualquier otro filtro en ActionScript. Cuando se usa un sombreado como filtro, la imagen filtrada (un objeto de visualización u objeto BitmapData) se transmite al sombreado. El sombreado utiliza la imagen de entrada para crear la salida del filtro, que suele ser una versión modificada de la imagen original. Si el objeto filtrado es un objeto de visualización, la salida del sombreado se muestra en pantalla en lugar del objeto de visualización filtrado. Si el objeto filtrado es un objeto BitmapData, la salida del sombreado se convierte en el contenido del objeto BitmapData cuyo método `applyFilter()` se llama.

Para utilizar un sombreado como filtro, en primer lugar cree el objeto Shader tal y como se describe en "Carga e incorporación de un sombreado" en la página 397. A continuación, cree un objeto ShaderFilter vinculado al objeto Shader. El objeto ShaderFilter es el filtro que se aplica al objeto filtrado. Se aplica a un objeto del mismo modo que se aplica cualquier filtro. Se transmite a la propiedad `filters` de un objeto de visualización o se llama al método `applyFilter()` en un objeto BitmapData. Por ejemplo, el siguiente código crea un objeto ShaderFilter y aplica el filtro a un objeto de visualización denominado `homeButton`.

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
homeButton.filters = [myFilter];
```

Si se utiliza una sombreado como filtro, el sombreado se debe definir con una entrada como mínimo. Tal y como muestra el ejemplo, en el código no se establece el valor de entrada. Sin embargo, el objeto de visualización filtrado u objeto BitmapData se define como imagen de entrada. Si se emplea un sombreado que espera varias entradas, indique un valor para todas las entradas que no sean la primera.

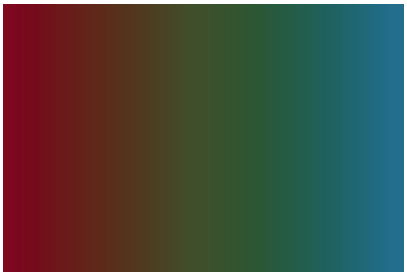
En algunos casos, un filtro cambia las dimensiones de la imagen original. Por ejemplo, un efecto típico de sombra paralela añade píxeles adicionales que contienen la sombra que se agrega a la imagen. Si se utiliza un sombreado que cambia las dimensiones de la imagen, establezca las propiedades `leftExtension`, `rightExtension`, `topExtension` y `bottomExtension` para indicar el grado de cambio de tamaño deseado de la imagen.

El siguiente ejemplo muestra el uso de un sombreado como filtro. El filtro de este ejemplo invierte los valores del canal rojo, verde y azul de una imagen. El resultado en la versión "negativa" de la imagen.

Nota: *el sombreado que utiliza este ejemplo es el núcleo de Pixel Bender invertRGB.pbk que se incluye en el kit de herramientas del lenguaje. Puede cargar el código fuente para el núcleo desde el directorio de instalación del kit de herramientas de Pixel Bender. Compile el código fuente y guarde el archivo de código de bytes en el mismo directorio que el código fuente.*

El código de ActionScript importante se encuentra en estos dos métodos:

- `init()`: el método `init()` se llama cuando se carga la aplicación. En este método el código carga el archivo de código de bytes de sombreado.
- `onLoadComplete()`: en el método `onLoadComplete()` el código crea el objeto Shader denominado `shader`. Crea y dibuja el contenido de un objeto denominado `target`. El objeto `target` es un rectángulo relleno con un color de degradado lineal que es rojo en la parte izquierda, amarillo-verde en el centro y azul claro en la parte derecha. El objeto no filtrado presenta el siguiente aspecto:



Cuando se aplica el filtro los colores se invierten y el rectángulo presenta el siguiente aspecto:

El sombreado que utiliza este ejemplo es el núcleo de Pixel Bender de ejemplo "invertRGB.pbk", incluido en el kit de herramientas del lenguaje. El código fuente está disponible en el archivo "invertRGB.pbk" en el directorio de instalación del kit de herramientas de Pixel Bender. Compile el código fuente y guarde el archivo de código de bytes con el nombre "invertRGB.pbj" en el mismo directorio que el código fuente de ActionScript.

Este es el código de ActionScript de este ejemplo. Utilice esta clase como clase principal de aplicación para un proyecto sólo ActionScript en Flex, o bien, como la clase de documento para el archivo FLA en la herramienta de edición de Flash:

```
package
{
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.filters.ShaderFilter;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class InvertRGB extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function InvertRGB()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("invertRGB.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
```

```
            shader = new Shader(loader.data);

            var target:Shape = new Shape();
            addChild(target);

            var g:Graphics = target.graphics;
            var c:Array = [0x990000, 0x445500, 0x007799];
            var a:Array = [255, 255, 255];
            var r:Array = [0, 127, 255];
            var m:Matrix = new Matrix();
            m.createGradientBox(w, h);
            g.beginGradientFill(GradientType.LINEAR, c, a, r, m);
            g.drawRect(10, 10, w, h);
            g.endFill();

            var invertFilter:ShaderFilter = new ShaderFilter(shader);
            target.filters = [invertFilter];
        }
    }
}
```

Para obtener más información sobre la aplicación de filtros, consulte “Creación y aplicación de filtros” en la página 364.

Utilización de un sombreado en modo autónomo

Cuando se utiliza un sombreado en modo autónomo, el procesamiento del sombreado se ejecuta independientemente de cómo se vaya a utilizar la salida. Se especifica un sombreado para ejecutar, se establecen los valores de parámetro y entrada y se designa un objeto en el que se sitúan los datos del resultado. Un sombreado se puede emplear en modo autónomo por dos motivos:

- Procesamiento de datos que no son de imagen: en el modo autónomo, puede optar por transmitir datos numéricos o binarios arbitrarios al sombreado en lugar de datos de imagen de mapa de bits. Puede optar por que el resultado del sombreado se devuelva como datos binarios o numéricos además de como datos de imagen de mapa de bits.
- Procesamiento en segundo plano: cuando un sombreado se ejecuta en modo autónomo, de forma predeterminada se hace asíncronicamente. Esto significa que el sombreado se ejecuta en segundo plano mientras la aplicación continúa en ejecución y el código se notifica cuando finaliza el procesamiento del sombreado. Puede utilizar un sombreado que tarde más tiempo en ejecutarse y no bloquee la interfaz de usuario de la aplicación u otro procesamiento mientras que el sombreado se esté ejecutando.

Puede utilizar un objeto ShaderJob para ejecuta un sombreado en modo autónomo. En primer lugar cree un objeto ShaderJob y vincúlelo al objeto Shader que representa el sombreado que se va a ejecutar:

```
var job:ShaderJob = new ShaderJob(myShader);
```

A continuación, se establecen todos los valores de parámetro o entrada que espera el sombreado. Si está ejecutando el sombreado en segundo plano, también se registra un detector para el evento `complete` del objeto ShaderJob. El detector se llama cuando el sombreado finaliza su tarea:

```
function completeHandler(event:ShaderEvent):void
{
    // do something with the shader result
}

job.addEventListener(ShaderEvent.COMPLETE, completeHandler);
```

A continuación, se crea un objeto en el que se escribe el resultado de la operación de sombreado cuando finaliza la operación. Ese objeto se asigna a la propiedad `target` del objeto ShaderJob:

```
var jobResult:BitmapData = new BitmapData(100, 75);
job.target = jobResult;
```

Asigne una instancia de BitmapData a la propiedad `target` si está utilizando ShaderJob para realizar el procesamiento de imagen. Si va a procesar datos numéricos o binarios, asigne un objeto ByteArray o instancia de Vector.<Number> a la propiedad `target`. En este caso, debe establecer las propiedades `width` y `height` del objeto ShaderJob para que especifiquen la cantidad de datos de salida en el objeto `target`.

Nota: *se pueden establecer las propiedades `shader`, `target`,`width` y `height` del objeto ShaderJob en un paso transmitiendo argumentos al constructor `ShaderJob()`, del siguiente modo:`var job:ShaderJob = new ShaderJob(myShader, myTarget, myWidth, myHeight);`*

Cuando esté listo para ejecutar el sombreado, puede llamar al método `start()` del objeto ShaderJob:

```
job.start();
```

De forma predeterminada, la llamada a `start()` hace que ShaderJob se ejecute asincrónicamente. En ese caso, la ejecución del programa continúa inmediatamente con la siguiente línea de código en lugar de esperar a que termine el sombreado. Una vez terminada la operación del sombreado, el objeto ShaderJob llama a sus detectores de eventos `complete`, notificándoles que se ha finalizado. En este punto (es decir, en el cuerpo del detector de eventos `complete`) el objeto `target` contiene el resultado de la operación de sombreado.

Nota: *en lugar de utilizar el objeto de propiedad `target`, puede recuperar el resultado del sombreado directamente del objeto de evento que se transmite al método detector. El objeto de evento es una instancia de ShaderEvent. El objeto ShaderEvent tiene tres propiedades que se pueden utilizar para acceder al resultado, dependiendo del tipo de datos del objeto que se establezca como propiedad `target`: `ShaderEvent.bitmapData`, `ShaderEvent.byteArray` y `ShaderEvent.vector`.*

De forma alternativa, puede pasar un argumento `true` al método `start()`. En ese caso, la operación de sombreado se ejecuta sincrónicamente. Todo el código (incluyendo la interacción con la interfaz de usuario y cualquier otro evento) se detiene mientras se ejecuta el sombreado. Cuando el sombreado finaliza, el objeto `target` contiene el resultado del proceso y el programa continúa con la siguiente línea de código.

```
job.start(true);
```

Capítulo 18: Trabajo con clips de película

La clase MovieClip es la clase principal para animación y símbolos de clip de película creados en Adobe® Flash® CS4 Professional. Tiene todos los comportamientos y la funcionalidad de los objetos de visualización, pero con propiedades y métodos adicionales para controlar la línea de tiempo de un clip de película. En este capítulo se explica la manera de utilizar ActionScript para controlar la reproducción de un clip de película y para crear dinámicamente un clip de película.

Fundamentos de la utilización de película

Introducción a la utilización de clips de película

Los clips de película son un elemento clave para las personas que crean contenido animado con la herramienta de edición Flash y desean controlar el contenido con ActionScript. Siempre que se crea un símbolo de clip de película en Flash, Flash añade el símbolo a la biblioteca del documento Flash en el que se crea. De forma predeterminada, este símbolo se convierte en una instancia de la clase MovieClip y, como tal, tiene las propiedades y los métodos de la clase MovieClip.

Cuando se coloca una instancia de un símbolo de clip de película en el escenario, el clip de película progresa automáticamente por su línea de tiempo (si tiene más de un fotograma) a menos que se haya modificado la reproducción con ActionScript. Esta línea de tiempo es lo que distingue a la clase MovieClip, que permite crear animación mediante interpolaciones de movimiento y de forma, a través de la herramienta de edición de Flash. En cambio, con un objeto de visualización que es una instancia de la clase Sprite sólo se puede crear una animación modificando mediante programación los valores del objeto.

En versiones anteriores de ActionScript, la clase MovieClip era la clase base de todas las instancias en el escenario. En ActionScript 3.0, un clip de película sólo es uno de los numerosos objetos de visualización que aparecen en la pantalla. Si no se necesita una línea de tiempo para el funcionamiento de un objeto de visualización, utilizar la clase Shape o Sprite en lugar de la clase MovieClip puede mejorar el rendimiento de la representación. Para obtener más información sobre cómo elegir el objeto de visualización apropiado para una tarea, consulte “Selección de una subclase DisplayObject” en la página 299.

Tareas comunes con clips de película

En este capítulo se describen las siguientes tareas comunes relacionadas con los clips de película:

- Reproducir y detener clips de película
- Reproducir clips de película hacia atrás
- Mover la cabeza lectora a puntos específicos de una línea de tiempo de clip de película
- Trabajo con etiquetas de fotogramas en ActionScript
- Acceder a información de escenas en ActionScript
- Crear instancias de símbolos de clip de película de bibliotecas mediante ActionScript
- Cargar y controlar archivos SWF externos, incluidos archivos creados para versiones anteriores de Flash Player
- Generar un sistema ActionScript para crear elementos gráficos que deban cargarse y utilizarse en tiempo de ejecución

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- AVM1 SWF: archivo SWF creado con ActionScript 1.0 o ActionScript 2.0 para Flash Player 8 o una versión anterior.
- AVM2 SWF: archivo SWF creado con ActionScript 3.0 para Adobe Flash Player 9 (o versión posterior) o Adobe AIR.
- Archivo SWF externo: archivo SWF que se crea de forma independiente del archivo SWF del proyecto y que debe cargarse en el archivo SWF del proyecto y reproducirse en dicho archivo SWF.
- Fotograma: división de tiempo más pequeña en la línea de tiempo. Al igual que en una tira de película animada, cada fotograma es como una instantánea de la animación en el tiempo; cuando se reproduce rápidamente la secuencia de los fotogramas, se crea el efecto de animación.
- Línea de tiempo: representación metafórica de una serie de fotogramas que forman una secuencia de animación de un clip de película. La línea de tiempo de un objeto MovieClip es equivalente a la línea de tiempo de la herramienta de edición Flash.
- Cabeza lectora: marcador que identifica la ubicación (fotograma) en la línea de tiempo que se está mostrando en un momento determinado.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Como este capítulo se centra en la utilización de clips de película en ActionScript, prácticamente todos los listados de código que contiene se han escrito con la idea de manipular un símbolo de clip de película creado y colocado en el escenario. Para probar el ejemplo es necesario ver el resultado en Flash Player o AIR para poder ver los efectos del código en el símbolo. Para probar los listados de código de este capítulo:

1. Cree un documento de Flash vacío.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Cree una instancia de símbolo de clip de película en el escenario. Por ejemplo, dibuje una forma, selecciónela, elija Modificar > Convertir en símbolo y asigne un nombre al símbolo.
5. Con el clip de película seleccionado, asígnele un nombre de instancia en el inspector de propiedades. El nombre debe coincidir con el utilizado para el clip de película en el listado de código de ejemplo; por ejemplo, si el listado de código manipula un clip de película denominado `myMovieClip`, debe asignar un nombre a la instancia del clip `myMovieClip`.
6. Ejecute el programa seleccionando Control > Probar película.

   Verá en pantalla el resultado de la manipulación del clip de película realizada por el código.

En el capítulo "Prueba de los listados de código de ejemplo del capítulo" en la página 36 se explican de forma más detallada otra técnicas para la comprobación de listados de código.

Trabajo con objetos MovieClip

Cuando se publica un archivo SWF, Flash convierte de forma predeterminada todas las instancias de símbolo de clip de película del escenario en objetos MovieClip. Para hacer que un símbolo de clip de película esté disponible para ActionScript, se le asigna un nombre de instancia en el campo Nombre de instancia del inspector de propiedades. Cuando se crea el archivo SWF, Flash genera el código que crea la instancia de MovieClip en el escenario y declara una variable con el nombre de instancia. Si se tienen clips de película con nombre anidados en otros clips de película con nombre, los clips de película secundarios se tratarán como propiedades del clip de película principal (se puede acceder al clip de película secundario con la sintaxis de punto). Por ejemplo, si un clip de película con el nombre de la instancia `childClip` está anidado dentro de otro clip con el nombre de la instancia `parentClip`, se puede hacer que se reproduzca la animación de la línea de tiempo del clip secundario llamando a este código:

```
parentClip.childClip.play();
```

Nota: *no se puede acceder a las instancias secundarias colocadas en el escenario en la herramienta de edición de Flash por código del constructor de una instancia principal, ya que no se han creado en ese punto en la ejecución del código. Antes de acceder al valor secundario, el principal debe crear la instancia secundaria por código o acceder con retraso a la función callback que detecta el valor secundario para que distribuya el evento* `Event.ADDED_TO_STAGE`*.*

Aunque se conservan algunos métodos y propiedades de la clase MovieClip de ActionScript 2.0, otros han cambiado. Todas las propiedades que empiezan por un carácter de subrayado han cambiado de nombre. Por ejemplo, las propiedades `_width` y `_height` son ahora `width` y `height`, mientras que las propiedades `_xscale` y `_yscale` son ahora `scaleX` y `scaleY`. Para ver una lista completa de las propiedades y métodos de la clase MovieClip, consulte Referencia del lenguaje y componentes ActionScript 3.0.

Control de la reproducción de clips de película

Flash utiliza la metáfora de una línea de tiempo para expresar animación o un cambio de estado. Cualquier elemento visual que emplee una línea de tiempo debe ser un objeto MovieClip o una ampliación de la clase MovieClip. Aunque se puede utilizar el código ActionScript para ordenar a cualquier clip de película que se detenga, se reproduzca o pase a otro punto de la línea de tiempo, no se puede utilizar para crear dinámicamente una línea de tiempo ni añadir contenido en fotogramas específicos; esto sólo es posible en la herramienta de edición Flash.

Cuando se reproduce un clip de película, avanza por su línea de tiempo a la velocidad indicada en la velocidad de fotogramas del archivo SWF. Como alternativa, se puede sustituir esta configuración estableciendo la propiedad `Stage.frameRate` en ActionScript.

Reproducción de clips de película y detención de la reproducción

Los métodos `play()` y `stop()` permiten realizar el control básico de un clip de película a lo largo de su línea de tiempo. Por ejemplo, sea un símbolo de clip de película en el escenario que contiene una animación de una bicicleta moviéndose por la pantalla, con el nombre de instancia establecido en `bicycle`. Si se añade el código siguiente a un fotograma clave de la línea de tiempo principal,

```
bicycle.stop();
```

la bicicleta no se moverá (su animación no se reproducirá). El movimiento de la bicicleta puede empezar a través de alguna interacción del usuario. Por ejemplo, si hubiera un botón denominado `startButton`, el código siguiente en un fotograma clave de la línea de tiempo principal lo haría de forma que al hacer clic en el botón se reproduce la animación:

```
// This function will be called when the button is clicked. It causes the
// bicycle animation to play.
function playAnimation(event:MouseEvent):void
{
    bicycle.play();
}
// Register the function as a listener with the button.
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

Avance rápido y rebobinado

Los métodos `play()` y `stop()` no son la única manera de controlar la reproducción en un clip de película. También se puede avanzar o rebobinar manualmente la cabeza lectora a lo largo de la línea de tiempo con los métodos `nextFrame()` y `prevFrame()`. Cuando se llama a cualquiera de estos métodos, se detiene la reproducción, y la cabeza lectora avanza o rebobina un fotograma, respectivamente.

Utilizar el método `play()` equivale a llamar a `nextFrame()` cada vez que se activa el evento `enterFrame` del objeto de clip de película. De esta manera, se podría reproducir hacia atrás el clip de película `bicycle` añadiendo un detector de eventos para el evento `enterFrame` e indicando a `bicycle` que retroceda al fotograma anterior en la función de detector, como se muestra a continuación:

```
// This function is called when the enterFrame event is triggered, meaning
// it's called once per frame.
function everyFrame(event:Event):void
{
    if (bicycle.currentFrame == 1)
    {
        bicycle.gotoAndStop(bicycle.totalFrames);
    }
    else
    {
        bicycle.prevFrame();
    }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

En la reproducción normal, si un clip de película contiene más de un fotograma, se reproducirá indefinidamente; es decir, que volverá al fotograma 1 si avanza más allá del último fotograma. Al utilizar `prevFrame()` o `nextFrame()`, este comportamiento no se produce automáticamente (llamar a `prevFrame()` cuando la cabeza lectora está en el Fotograma 1 no mueve la cabeza lectora al último fotograma). La condición `if` del ejemplo anterior comprueba si la cabeza lectora ha retrocedido al primer fotograma y la avanza hasta el último fotograma, creando un bucle continuo del clip de película que se reproduce hacia atrás.

Salto a otro fotograma y utilización de etiquetas de fotogramas

Enviar un clip de película a un nuevo fotograma es muy sencillo. Mediante una llamada a `gotoAndPlay()` o `gotoAndStop()`, el clip de película saltará al número de fotograma especificado como parámetro. Como alternativa, se puede pasar una cadena que coincida con el nombre de una etiqueta de fotograma. Se puede asignar una etiqueta a cualquier fotograma de la línea de tiempo. Para ello, hay que seleccionar un fotograma de la línea de tiempo e introducir un nombre en el campo Etiqueta de fotograma del inspector de propiedades.

Las ventajas de utilizar etiquetas de fotogramas en lugar de números se aprecian especialmente cuando se crea un clip de película complejo. Cuando el número de fotogramas, capas e interpolaciones de una animación es elevado, resulta útil etiquetar los fotogramas importantes con descripciones explicativas que representan los cambios en el comportamiento del clip de película; por ejemplo, "off" (fuera), "walking" (caminando) o "running" (corriendo). De esta forma se mejora la legibilidad del código y se logra una mayor flexibilidad, ya que las llamadas de código ActionScript para ir a un fotograma con etiqueta señalan a una sola referencia, la etiqueta, en lugar de a un número de fotograma específico. Si posteriormente se decide mover la cabeza lectora a un segmento específico de la animación en un fotograma diferente, será necesario modificar el código ActionScript manteniendo la misma etiqueta para los fotogramas en la nueva ubicación.

Para representar etiquetas de fotograma en el código, ActionScript 3.0 incluye la clase FrameLabel. Cada instancia de esta clase representa una sola etiqueta de fotograma y tiene una propiedad `name`, que representa el nombre de la etiqueta de fotograma especificado en el inspector de propiedades, y una propiedad `frame` que representa el número de fotograma del fotograma en el que está colocada la etiqueta en la línea de tiempo.

Para acceder a las instancias de FrameLabel asociadas con una instancia de clip de película, la clase MovieClip incluye dos propiedades que devuelven objetos FrameLabel directamente. La propiedad `currentLabels` devuelve un conjunto formado por todos los objetos FrameLabel de toda la línea de tiempo de un clip de película. La propiedad `currentLabel` devuelve una cadena que contiene el nombre de la etiqueta de fotograma encontrada más reciente en la línea de tiempo.

Supongamos que se crea un clip de película denominado `robot` y que se asignan etiquetas a los distintos estados de animación. Se podría configurar una condición que comprobara la etiqueta `currentLabel` para acceder al estado actual del robot, como en el código siguiente:

```
if (robot.currentLabel == "walking")
{
    // do something
}
```

Trabajo con escenas

En el entorno de edición de Flash, se pueden utilizar las escenas para delimitar una serie de líneas de tiempo en las que avanzaría un archivo SWF. En el segundo parámetro de los métodos `gotoAndPlay()` o `gotoAndStop()`, se puede especificar una escena a la que enviar la cabeza lectora. Todos los archivos FLA comienzan con la escena inicial únicamente, pero se pueden crear nuevas escenas.

La utilización de escenas no es siempre el mejor enfoque, ya que presenta varios inconvenientes. Un documento de Flash que contenga varias escenas puede ser difícil de mantener, especialmente en entornos de varios autores. La utilización de varias escenas puede no ser eficaz en términos de ancho de banda, ya que el proceso de publicación combina todas las escenas en una sola línea de tiempo. Esto provoca una descarga progresiva de todas las escenas, incluso si no se reproducen nunca. Por estos motivos, a menudo se desaconseja utilizar varias escenas, a menos que se necesiten para organizar varias animaciones largas basadas en la línea de tiempo.

La propiedad `scenes` de la clase MovieClip devuelve un conjunto de objetos Scene que representan todas las escenas del archivo SWF. La propiedad `currentScene` devuelve un objeto Scene que representa la escena que se está reproduciendo actualmente.

La clase Scene tiene varias propiedades que ofrecen información sobre una escena. La propiedad `labels` devuelve un conjunto de objetos FrameLabel que representan las etiquetas de fotograma en dicha escena. La propiedad `name` devuelve el nombre de la escena como una cadena. La propiedad `numFrames` devuelve un entero que representa el número total de fotogramas en la escena.

Creación de objetos MovieClip con ActionScript

Una forma de añadir contenido a la pantalla en Flash consiste en arrastrar los activos de la biblioteca al escenario. Sin embargo, no es el único modo. En proyectos completos, los desarrolladores con experiencia suelen preferir crear clips de película mediante programación. Este enfoque implica varias ventajas: reutilización de código más sencilla, velocidad de tiempo de compilación más rápida y modificaciones más sofisticadas que sólo están disponibles en ActionScript.

La API de lista de visualización de ActionScript 3.0 optimiza el proceso de creación dinámica de objetos MovieClip. La capacidad de crear una instancia de MovieClip directamente, por separado del proceso de añadirla a la lista de visualización, proporciona flexibilidad y simplicidad sin sacrificar control.

En ActionScript 3.0, al crear una instancia de clip de película (o cualquier otro objeto de visualización) mediante programación, no estará visible en la pantalla hasta que se añada a la lista de visualización llamando al método `addChild()` o `addChildAt()` en un contenedor de objeto de visualización. Esto permite crear un clip de película, establecer sus propiedades e incluso llamar a métodos antes de representarlo en la pantalla. Para obtener más información sobre la utilización de la lista de visualización, consulte "Trabajo con contenedores de objetos de visualización" en la página 287.

Exportación de símbolos de biblioteca para ActionScript

De manera predeterminada, las instancias de símbolos de clip de película de una biblioteca de un documento Flash no se pueden crear dinámicamente (es decir, mediante ActionScript). El motivo de ello es que cada símbolo que se exporta para utilizarlo en ActionScript se suma al tamaño del archivo SWF y se detecta que no todos los símbolos se van a utilizar en el escenario. Por esta razón, para que un símbolo esté disponible en ActionScript, hay que especificar que se debe exportar el símbolo para ActionScript.

Para exportar un símbolo para ActionScript:

1. Seleccione el símbolo en el panel Biblioteca y abra su cuadro de diálogo Propiedades de símbolo.
2. Si es necesario, active la configuración avanzada.
3. En la sección Vinculación, active la casilla de verificación Exportar para ActionScript.

   De este modo, se activan los campos Clase y Clase base.

   De manera predeterminada, el campo Clase se llena con el nombre del símbolo, eliminando los espacios (por ejemplo, un símbolo denominado "Tree House" se convertiría en "TreeHouse"). Para especificar que el símbolo debe utilizar una clase personalizada para su comportamiento, hay que escribir el nombre completo de la clase, incluido el paquete, en este campo. Si se desea crear instancias del símbolo en ActionScript y no es necesario añadir ningún comportamiento adicional, se puede dejar el nombre de la clase tal cual.

   El valor predeterminado del campo Clase base es `flash.display.MovieClip`. Si se desea que el símbolo amplíe la funcionalidad de otra clase de cliente, se puede especificar el nombre de la clase, siempre y cuando dicha clase amplíe la clase Sprite (o MovieClip).
4. Presione el botón Aceptar para guardar los cambios.

   Si Flash no encuentra un archivo de ActionScript externo con una definición para la clase especificada (por ejemplo, si no se necesita un comportamiento adicional para el símbolo), se muestra una advertencia:

   No se pudo encontrar una definición de esta clase en la ruta de clases, por lo que se generará una automáticamente en el archivo SWF al exportar.

Se puede pasar por alto esta advertencia si el símbolo de la biblioteca no requiere funcionalidad exclusiva más allá de la funcionalidad de la clase MovieClip.

Si no se proporciona una clase para el símbolo, Flash creará una clase para el símbolo equivalente a la siguiente:

```
package
{
    import flash.display.MovieClip;

    public class ExampleMovieClip extends MovieClip
    {
        public function ExampleMovieClip()
        {
        }
    }
}
```

Si se desea añadir funcionalidad de ActionScript adicional al símbolo, hay que añadir las propiedades y métodos adecuados a la estructura de código. Por ejemplo, sea un símbolo de clip de película que contiene un círculo con una anchura de 50 píxeles y una altura de 50 píxeles, y se especifica que el símbolo debe exportarse para ActionScript con una clase denominada Circle. El siguiente código, incluido en un archivo Circle.as, amplía la clase MovieClip y proporciona al símbolo los métodos adicionales `getArea()` y `getCircumference()`:

```
package
{
    import flash.display.MovieClip;

    public class Circle extends MovieClip
    {
        public function Circle()
        {
        }

        public function getArea():Number
        {
        // The formula is Pi times the radius squared.
        return Math.PI * Math.pow((width / 2), 2);
        }

        public function getCircumference():Number
        {
        // The formula is Pi times the diameter.
        return Math.PI * width;
        }
    }
}
```

El código siguiente, colocado en un fotograma clave en el Fotograma 1 del documento de Flash, creará una instancia del símbolo y la mostrará en pantalla:

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

Este código muestra la creación de instancias basada en ActionScript como una alternativa a arrastrar activos individuales al escenario. Crea un círculo que tiene todas las propiedades de un clip de película, además de los métodos personalizados definidos en la clase Circle. Es un ejemplo muy básico: el símbolo de la biblioteca puede especificar varias propiedades y métodos en su clase.

La creación de instancias basada en ActionScript es eficaz, ya que permite crear de forma dinámica grandes cantidades de instancias, una tarea tediosa si tuviera que realizarse manualmente. También es flexible, pues permite personalizar las propiedades de cada instancia a medida que se crea. Para comprobar ambas ventajas, utilice un bucle para crear de forma dinámica varias instancias de Circle. Con el símbolo y la clase Circle descritos previamente en la biblioteca del documento Flash, coloque el código siguiente en un fotograma clave en el Fotograma 1:

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
    // Create a new Circle instance.
    var c:Circle = new Circle();
    // Place the new Circle at an x coordinate that will space the circles
    // evenly across the Stage.
    c.x = (stage.stageWidth / totalCircles) * i;
    // Place the Circle instance at the vertical center of the Stage.
    c.y = stage.stageHeight / 2;
    // Change the Circle instance to a random color
    c.transform.colorTransform = getRandomColor();
    // Add the Circle instance to the current timeline.
    addChild(c);
}

function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Esto ilustra la forma de crear y personalizar múltiples instancias de un símbolo rápidamente mediante código. La posición de cada instancia se modifica en función del recuento actual dentro del bucle y cada instancia recibe un color aleatorio mediante la propiedad `transform` (que hereda Circle a través de la ampliación de la clase MovieClip).

Carga de un archivo SWF externo

En ActionScript 3.0, los archivos SWF se cargan mediante la clase Loader. Para cargar un archivo SWF externo, el código ActionScript debe hacer cuatro cosas:

1. Crear un nuevo objeto URLRequest con el URL del archivo.
2. Crear un nuevo objeto Loader.
3. Llamar al método `load()` del objeto Loader pasando la instancia de URLRequest como parámetro.

4 Llamar al método `addChild()` en un contendor de objeto de visualización (como la línea de tiempo principal de un documento de Flash) para añadir la instancia de Loader a la lista de visualización.

Una vez finalizado, el código tiene el siguiente aspecto:

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

Este mismo código se puede utilizar para cargar un archivo de imagen externo como una imagen JPEG, GIF o PNG especificando el URL del archivo de imagen en lugar del URL del archivo SWF. Un archivo SWF, a diferencia de un archivo de imagen, puede contener código ActionScript. Por lo tanto, aunque el proceso de carga de un archivo SWF puede ser idéntico a la carga de una imagen, al cargar un archivo SWF externo, tanto el SWF que realiza la carga como el SWF cargado deben residir en el mismo entorno limitado de seguridad si Flash Player o AIR van a reproducir el archivo SWF y ActionScript se va a emplear para comunicarse de cualquier forma con el archivo SWF externo. Además, si el archivo SWF externo contiene clases que comparten el mismo espacio de nombres que las clases del archivo SWF que realiza la carga, es posible que sea necesario crear un nuevo dominio de aplicación para el archivo SWF cargado a fin de evitar conflictos de espacio de nombres. Para obtener más información acerca de las consideraciones sobre seguridad y el dominio de aplicación, consulte "Utilización de la clase ApplicationDomain" en la página 667 y "Carga de archivos SWF e imágenes" en la página 729.

Cuando el archivo SWF externo se carga correctamente, se puede acceder a él a través de la propiedad `Loader.content`. Si el archivo SWF externo se publica para ActionScript 3.0, será un objeto MovieClip o Sprite, en función de la clase que amplíe.

Consideraciones sobre la carga de un archivo SWF antiguo

Si el archivo SWF externo se ha publicado con una versión anterior de ActionScript, hay que tener en cuenta algunas limitaciones importantes. A diferencia de un archivo SWF de ActionScript 3.0 que se ejecuta en AVM2 (máquina virtual ActionScript 2), un archivo SWF publicado para ActionScript 1.0 ó 2.0 se ejecuta en AVM1 (máquina virtual ActionScript 1).

Cuando un archivo SWF AVM1 se carga correctamente, el objeto cargado (la propiedad `Loader.content`) será un objeto AVM1Movie. Una instancia de AVM1Movie no es lo mismo que una instancia de MovieClip. Es un objeto de visualización que, a diferencia de un clip de película, no incluye propiedades ni métodos relacionados con la línea de tiempo. El archivo SWF AVM2 principal no tendrá acceso a las propiedades, métodos ni objetos del objeto AVM1Movie cargado.

Existen restricciones adicionales en un archivo SWF AVM1 cargado por un archivo SWF AVM2: Para obtener información, consulte el listado de clases AVM1Movie en la Referencia del lenguaje y componentes ActionScript 3.0.

Ejemplo: RuntimeAssetsExplorer

La funcionalidad Exportar para ActionScript puede ser especialmente ventajosa para las bibliotecas que pueden utilizarse en más de un proyecto. Si Flash Player o AIR ejecutan un archivo SWF, los símbolos que se han exportado a ActionScript están disponibles en cualquier archivo SWF del mismo entorno limitado de seguridad que el archivo SWF que lo carga. De este modo, un solo documento de Flash puede generar un archivo SWF cuyo único propósito sea contener activos gráficos. Esta técnica es especialmente útil en proyectos grandes donde los diseñadores que trabajan en activos visuales pueden trabajar en paralelo con los desarrolladores que crean un archivo SWF "envolvente", que carga los archivos SWF de activos gráficos en tiempo de ejecución. Puede utilizarse este método para mantener una serie de archivos de versiones cuyos activos gráficos no dependan del progreso de desarrollo de programación.

La aplicación RuntimeAssetsExplorer carga cualquier archivo SWF que sea una subclase de RuntimeAsset y permite examinar los activos disponibles de dicho archivo SWF. El ejemplo muestra lo siguiente:

- Carga de un archivo SWF mediante `Loader.load()`
- Creación dinámica de un símbolo de la biblioteca exportado para ActionScript
- Control de la reproducción de MovieClip en el código ActionScript

Antes de comenzar, se debe que tener en cuenta que los archivos SWF que se van a ejecutar en Flash Player se deben ubicar en el mismo entorno limitado de seguridad. Para obtener más información, consulte “Entornos limitados de seguridad” en la página 716.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación RuntimeAssetsExplorer se encuentran en la carpeta Samples/RuntimeAssetsExplorer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
RuntimeAssetsExample.mxml o RuntimeAssetsExample.fla	La interfaz de usuario de la aplicación para Flex (MXML) o Flash (FLA).
RuntimeAssetsExample.as	Clase de documento para la aplicación de Flash (FLA).
GeometricAssets.as	Una clase de ejemplo que implementa la interfaz de RuntimeAsset.
GeometricAssets.fla	Un archivo FLA vinculado a la clase GeometricAssets (la clase de documento del archivo FLA), que contiene símbolos exportados para ActionScript.
com/example/programmingas3/runtimeassetexplorer/RuntimeLibrary.as	Una interfaz que define los métodos necesarios que se esperan de todos los archivos SWF de activos de tiempo de ejecución que se cargarán en el contenedor del explorador.
com/example/programmingas3/runtimeassetexplorer/AnimatingBox.as	La clase del símbolo de la biblioteca con la forma de un cuadro giratorio.
com/example/programmingas3/runtimeassetexplorer/AnimatingStar.as	La clase del símbolo de la biblioteca con la forma de una estrella giratoria.

Establecimiento de una interfaz de biblioteca de tiempo de ejecución

Para que el explorador pueda interactuar correctamente con la biblioteca de un archivo SWF, debe formalizarse la estructura de las bibliotecas de activos de tiempo de ejecución. Con este fin se creará una interfaz, que se asemeja a una clase en que es un plano de los métodos que delimitan una estructura esperada, pero a diferencia de una clase no incluye el cuerpo de los métodos. La interfaz proporciona una forma de comunicación entre la biblioteca de tiempo de ejecución y el explorador. Cada archivo SWF de activos de tiempo de ejecución que se cargue en el navegador implementará esta interfaz. Para obtener más información sobre las interfaces y su utilidad, consulte"Interfaces" en la página 109.

La interfaz de RuntimeLibrary será muy sencilla; sólo se necesita una función que pueda proporcionar al explorador un conjunto de rutas de clases para los símbolos que se exportarán y estarán disponibles en la biblioteca de tiempo de ejecución. Para este fin, la interfaz tiene un solo método(): `getAssets()`.

```
package com.example.programmingas3.runtimeassetexplorer
{
    public interface RuntimeLibrary
    {
        function getAssets():Array;
    }
}
```

Creación del archivo SWF de biblioteca de activos

Mediante la definición de la interfaz de RuntimeLibrary, es posible crear varios archivos SWF de biblioteca de activos que se pueden cargar en otro archivo SWF. La creación de un archivo SWF de biblioteca de activos implica cuatro tareas:

- Crear una clase para el archivo SWF de biblioteca de activos
- Crear clases para activos individuales contenidos en la biblioteca
- Crear los activos gráficos reales
- Asociar los elementos gráficos a las clases y publicar el archivo SWF de biblioteca

Crear una clase para implementar la interfaz RuntimeLibrary

A continuación, se creará la clase GeometricAssets que implementará la interfaz de RuntimeLibrary. Ésta será la clase de documento del archivo FLA. El código para esta clase es muy similar a la interfaz RuntimeLibrary. Se diferencian en que en la definición de clase el método `getAssets()` no tiene cuerpo de método.

```
package
{
    import flash.display.Sprite;
    import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

    public class GeometricAssets extends Sprite implements RuntimeLibrary
    {
        public function GeometricAssets() {

        }
        public function getAssets():Array {
            return [ "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",
                     "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
        }
    }
}
```

Si se tuviera que crear una segunda biblioteca de tiempo de ejecución, se crearía otro archivo FLA basado en otra clase (por ejemplo, AnimationAssets) que proporcionara su propia implementación de `getAssets()`.

Creación de clases para cada activo de MovieClip

En este ejemplo, simplemente se amplía la clase MovieClip sin añadir ninguna funcionalidad a los activos personalizados. El siguiente código de AnimatingStar equivale al de AnimatingBox:

```
package com.example.programmingas3.runtimeassetexplorer
{
    import flash.display.MovieClip;

    public class AnimatingStar extends MovieClip
    {
        public function AnimatingStar() {
        }
    }
}
```

Publicación de la biblioteca

Ahora se conectarán los activos basados en MovieClip con la nueva clase; para ello, se creará un nuevo archivo FLA y se introducirá GeometricAssets en el campo Clase de documento del inspector de propiedades. Para este ejemplo, se crearán dos formas muy básicas que utilizan una interpolación de la línea de tiempo para girar en el sentido de las agujas del reloj a través de 360 fotogramas. Los símbolos `animatingBox` y `animatingStar` están configurados en Exportar para ActionScript y tienen como valor del campo Clase las rutas de clases respectivas especificadas en la implementación de `getAssets()`. La clase base predeterminada de `flash.display.MovieClip` se conserva, ya que se va a crear una subclase de los métodos estándar de MovieClip.

Después de establecer la configuración de exportación del símbolo, hay que publicar el archivo FLA para obtener la primera biblioteca de tiempo de ejecución. Este archivo SWF podría cargarse en otro archivo SWF AVM2, y los símbolos AnimatingBox y AnimatingStar estarían disponibles para el nuevo archivo SWF.

Carga de la biblioteca en otro archivo SWF

La última parte funcional que hay que resolver es la interfaz del usuario para el explorador de activos. En este ejemplo, la ruta a la biblioteca de tiempo de ejecución se especifica en el código como una variable denominada `ASSETS_PATH`. También se puede utilizar la clase FileReference, por ejemplo, para crear una interfaz que busque un determinado archivo SWF en el disco duro.

Cuando la biblioteca de tiempo de ejecución se carga correctamente, Flash Player llama al método `runtimeAssetsLoadComplete()`:

```
private function runtimeAssetsLoadComplete(event:Event):void
{
    var rl:* = event.target.content;
    var assetList:Array = rl.getAssets();
    populateDropdown(assetList);
    stage.frameRate = 60;
}
```

En este método, la variable rl representa el archivo SWF cargado. El código llama al método `getAssets()` del archivo SWF cargado, que obtiene la lista de activos disponibles, y los utiliza para llenar un componente ComboBox con una lista de activos disponibles llamando al método `populateDropDown()`. Este método almacena la ruta de clase completa de cada activo. Si se hace clic en el botón Añadir de la interfaz de usuario se activa el método `addAsset()`:

```
private function addAsset():void
{
    var className:String = assetNameCbo.selectedItem.data;
    var AssetClass:Class = getDefinitionByName(className) as Class;
    var mc:MovieClip = new AssetClass();
    ...
}
```

que obtiene la ruta de clase del activo que esté seleccionado actualmente en el ComboBox (`assetNameCbo.selectedItem.data`) y utiliza la función `getDefinitionByName()` (del paquete flash.utils) para obtener una referencia a la clase del activo para crear una nueva instancia de dicho activo.

Capítulo 19: Trabajo con interpolaciones de movimiento

En la sección "Animación de objetos" en la página 318 se describe cómo implementar animaciones con scripts en ActionScript.

En esta sección describiremos una técnica distinta para crear animaciones en: las interpolaciones de movimiento. Esta técnica permite crear movimiento estableciéndolo interactivamente en un archivo FLA con Adobe® Flash® CS4 Professional. Seguidamente, se puede utilizar el movimiento en la animación basada en ActionScript en tiempo de ejecución.

Flash CS4 genera automáticamente el código ActionScript que implementa la interpolación de movimiento y permite que el usuario pueda copiarlo y reutilizarlo.

Para poder crear interpolaciones de movimiento es preciso disponer de una licencia de Adobe Professional (Flash CS4).

Fundamentos de interpolaciones de movimiento

Introducción a interpolaciones de movimiento en ActionScript

Las interpolaciones de movimiento son una forma sencilla de crear animaciones.

Una interpolación de movimiento modifica las propiedades de los objetos de visualización (como la posición o la rotación) fotograma por fotograma. Una interpolación de movimiento también puede cambiar el aspecto de un objeto de visualización mientras se mueve aplicando diversos filtros y otras propiedades. Es posible crear la interpolación de movimiento interactivamente con Flash (que genera el código ActionScript para la interpolación de movimiento). Desde Flash, utilice el comando Copiar movimiento como ActionScript 3.0 para copiar el código ActionScript que creó la interpolación de movimiento. Posteriormente, puede reutilizar el código ActionScript para crear movimientos en su propia animación dinámica en tiempo de ejecución.

Consulte la sección Interpolaciones de movimiento en el manual *Utilización de Flash CS4 Professional* para obtener más información sobre la creación de interpolaciones de movimiento.

Tareas comunes de interpolaciones de movimiento

El código ActionScript generado automáticamente que implementa una interpolación de movimiento hace lo siguiente:

- Crea una instancia de un objeto de movimiento para la interpolación de movimiento.
- Establece la duración de la interpolación de movimiento
- Añade las propiedades a la interpolación de movimiento
- Añade filtros a la interpolación de movimiento
- Asocia la interpolación de movimiento con su objeto u objetos de visualización

Términos y conceptos importantes

A continuación se destaca un término importante que encontrará en este capítulo:

- Interpolación de movimiento: una construcción que genera fotogramas intermedios de un objeto de visualización en distintos estados y tiempos. Produce un efecto por el cual el primer estado evoluciona suavemente hacia el segundo. Se utiliza para mover un objeto de visualización en el escenario, así como para agrandarlo, encogerlo, girarlo, difuminarlo o cambiar su color con el paso del tiempo.

Copiar scripts de interpolación de movimiento

Una interpolación genera fotogramas intermedios que muestran un objeto de visualización en distintos estados en dos fotogramas diferentes de una línea de tiempo. Produce un efecto por el cual la imagen del primer fotograma evoluciona suavemente hasta la imagen del segundo fotograma. En una interpolación de movimiento, el cambio de aspecto suele implicar también un cambio de posición del objeto de visualización, creándose así el movimiento. Además de cambiar la posición del objeto de visualización, una interpolación de movimiento también puede girar, sesgar, cambiar de tamaño o aplicar filtros al objeto.

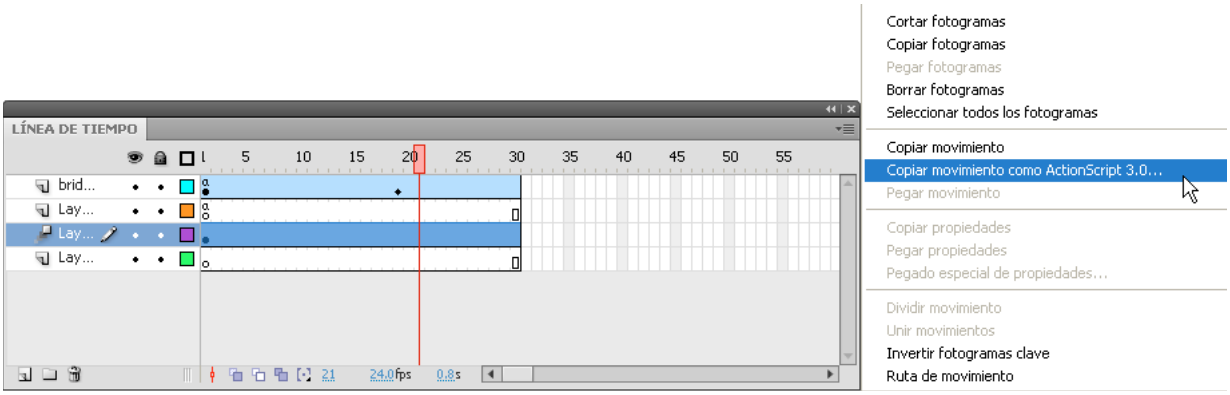
Para crear una interpolación de movimiento en Flash, mueva un objeto de visualización entre fotogramas clave de la línea de tiempo. Flash genera automáticamente el código ActionScript que describe la interpolación. Este código se puede copiar y guardar en un archivo. Consulte la sección Interpolaciones de movimiento en el manual *Utilización de Flash* para obtener más información sobre la creación de interpolaciones de movimiento.

Puede acceder al comando Copiar movimiento como ActionScript 3.0 en Flash de dos formas. La primera es desde el menú contextual de la interpolación en el escenario:

1 Seleccione la interpolación de movimiento en el escenario.

2 Haga clic con el botón derecho (Windows) o haga clic con la tecla Control pulsada (Macintosh).

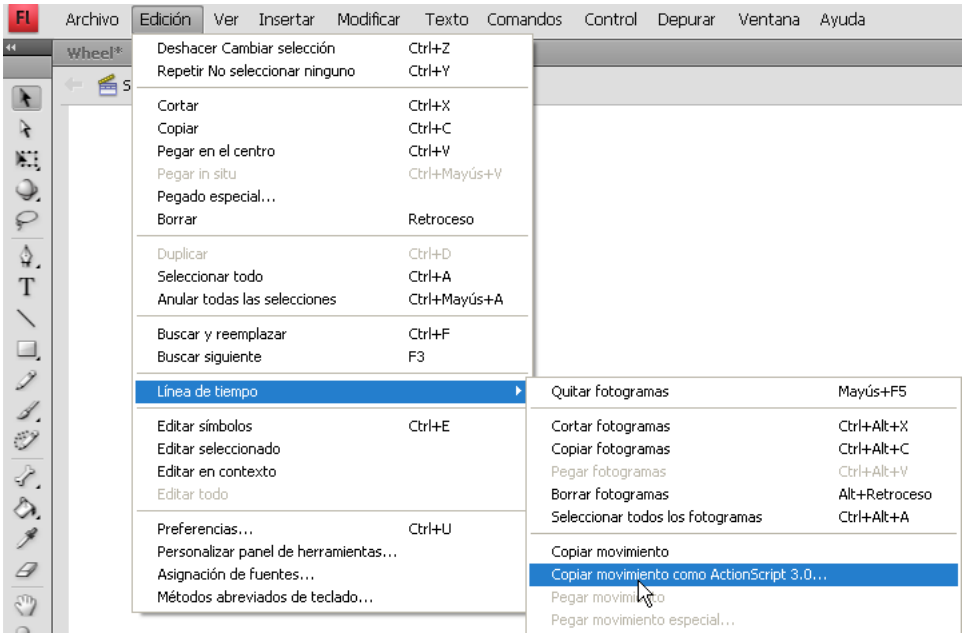
3 Seleccione Copiar movimiento como ActionScript 3.0...



La segunda posibilidad es elegir el comando directamente en el menú Edición de Flash

1 Seleccione la interpolación de movimiento en el escenario.

2 Seleccione Edición > Línea de tiempo > Copiar movimiento como ActionScript 3.0.



Una vez copiado el script, péguelo en un archivo y guárdelo.

Tras haber creado una interpolación de movimiento y haber copiado y guardado el script, puede reutilizarlo y modificarlo en su propia animación dinámica basada en ActionScript.

Incorporación de scripts de interpolación de movimiento

El encabezado del código ActionScript copiado desde Flash contiene los módulos necesarios para que se admita la interpolación de movimiento.

Clases de interpolación de movimiento

Las clases fundamentales son AnimatorFactory, MotionBase y Motion, pertenecientes al paquete `fl.motion`. Puede necesitar clases adicionales, dependiendo de las propiedades que manipule la interpolación de movimiento. Por ejemplo, si la interpolación de movimiento transforma o gira el objeto de visualización, deberá importar las clases adecuadas de `flash.geom`. Si aplica filtros, deberá importar las clases de `flash.filter`. En ActionScript, una interpolación de movimiento es una instancia de la clase Motion. La clase Motion guarda una secuencia de animación de fotogramas clave que se puede aplicar a un objeto visual. La información de animación incluye datos sobre ubicación, escala, rotación, sesgado, color, filtros y aceleración.

El siguiente código ActionScript se ha copiado de una interpolación de movimiento creada para animar un objeto de visualización con nombre de instancia `Symbol1_2`. Declara una variable para un objeto MotionBase llamada `__motion_Symbol1_2`. La clase MotionBase es el elemento principal de la clase Motion.

```
var __motion_Symbol1_2:MotionBase;
```

Seguidamente, el script crea el objeto Motion:

```
__motion_Symbol1_2 = new Motion();
```

Nombres de objetos Motion

En el caso anterior, Flash genera automáticamente el nombre `__motion_Symbol1_2` para el objeto Motion. Añade el prefijo `__motion_` al nombre del objeto de visualización. De este modo, el nombre generado automáticamente se basa en el nombre de instancia del objeto de destino de la interpolación de movimiento en la herramienta de edición de Flash. La propiedad `duration` del objeto Motion indica el número total de fotogramas en la interpolación de movimiento:

```
__motion_Symbol1_2.duration = 200;
```

Cuando reutilice este código ActionScript en su propia animación, puede conservar el nombre que Flash asigna automáticamente a la interpolación. También se puede utilizar un nombre distinto. De forma predeterminada, Flash asigna automáticamente un nombre a la instancia del objeto de visualización cuya interpolación de movimiento se está creando (si aún no tiene ningún nombre de instancia). Si cambia el nombre de la interpolación, debe asegurarse de que lo hace también en todo el script. Como alternativa, en Flash es posible asignar un nombre de su elección y asignarlo al objeto de destino de la interpolación de movimiento. Posteriormente, cree la interpolación de movimiento y copie el script. Puede utilizar la nomenclatura que desee, pero es importante que todos los objetos Motion del código ActionScript tengan un nombre exclusivo.

Descripción de la animación

El método `addPropertyArray()` de la clase MotionBase añade un conjunto de valores para describir cada propiedad interpolada.

Potencialmente, el conjunto contiene un elemento de conjunto en cada fotograma clave de la interpolación de movimiento. A menudo, algunos conjuntos contienen menos elementos que el número total de fotogramas clave de la interpolación de movimiento. Esta situación se produce cuando el último valor del conjunto no cambia en los fotogramas restantes.

Si la longitud del argumento del conjunto es superior a la propiedad `duration` del objeto Motion, `addPropertyArray()` ajusta el valor de la propiedad `duration` en consecuencia. No añade fotogramas clave a las propiedades añadidas previamente. Los fotogramas clave recién añadidos se conservan en los fotogramas adicionales de la animación.

Las propiedades `x` e `y` del objeto Motion describen la posición variable del objeto interpolado a medida que se ejecuta la animación. Estas coordenadas son los valores que, con toda probabilidad, cambiarán en cada fotograma clave si cambia la posición del objeto de visualización. Puede añadir propiedades de movimiento adicionales con el método `addPropertyArray()`. Por ejemplo, puede añadir los valores `scaleX` y `scaleY` si el objeto interpolado cambia de tamaño. Añada los valores `scewX` y `skewY` si se sesga. Añada la propiedad `rotationConcat` si se gira.

Utilice el método `addPropertyArray()` para definir las siguientes propiedades de interpolación:

`x`	posición horizontal del punto de transformación del objeto en el espacio de coordenadas de su elemento principal
`y`	posición vertical del punto de transformación del objeto en el espacio de coordenadas de su elemento principal
`z`	posición de profundidad (eje z) del punto de transformación del objeto en el espacio de coordenadas de su elemento principal
`scaleX`	escala horizontal (en forma de porcentaje) del objeto aplicada desde el punto de transformación
`scaleY`	escala vertical (en forma de porcentaje) del objeto aplicada desde el punto de transformación

`skewX`	ángulo de sesgado horizontal (en grados) del objeto de destino aplicado desde el punto de transformación
`skewY`	ángulo de sesgado vertical (en grados) del objeto de destino aplicado desde el punto de transformación
`rotationX`	rotación del objeto alrededor del eje x desde su orientación original
`rotationY`	rotación del objeto alrededor del eje y desde su orientación original
`rotationConcat`	valores de rotación (eje z) del objeto en el movimiento relativo a la orientación anterior (según se aplica desde el punto de transformación)
`useRotationConcat`	si se establece, hace que el objeto de destino gire cuando `addPropertyArray()` proporciona datos de movimiento
`blendMode`	valor de la clase BlendMode que especifica la mezcla de colores del objeto con gráficos subyacentes
`matrix3D`	propiedad matrix3D (si existe en el fotograma clave). Se utiliza para interpolaciones 3D; si se usa, todas las propiedades de transformación anteriores se omiten
`rotationZ`	rotación del objeto alrededor del eje z (en grados) desde su orientación original relativa al contendor principal 3D. Se utiliza en interpolaciones 3D para sustituir a rotationConcat

Las propiedades que se añaden en el script generado automáticamente dependen de las propiedades asignadas a la interpolación de movimiento en Flash. Es posible añadir, eliminar o modificar algunas de estas propiedades al personalizar su propia versión del script.

El siguiente código asigna valores a las propiedades de una interpolación de movimiento llamada `__motion_Wheel`. En este caso, el objeto de visualización interpolado no cambia de posición, sino que gira sobre sí mismo en los 29 fotogramas de la interpolación de movimiento. Los diversos valores asignados al conjunto `rotationConcat` definen la rotación. El resto de valores de las propiedades de la interpolación de movimiento no varían.

```
__motion_Wheel = new Motion();
__motion_Wheel.duration = 29;
__motion_Wheel.addPropertyArray("x", [0]);
__motion_Wheel.addPropertyArray("y", [0]);
__motion_Wheel.addPropertyArray("scaleX", [1.00]);
__motion_Wheel.addPropertyArray("scaleY", [1.00]);
__motion_Wheel.addPropertyArray("skewX", [0]);
__motion_Wheel.addPropertyArray("skewY", [0]);
__motion_Wheel.addPropertyArray("rotationConcat",
    [
        0,-13.2143,-26.4285,-39.6428,-52.8571,-66.0714,-79.2857,-92.4999,-105.714,
        -118.929,-132.143,-145.357,-158.571,-171.786,-185,-198.214,-211.429,-224.643,
        -237.857,-251.071,-264.286,-277.5,-290.714,-303.929,-317.143,-330.357,
        -343.571,-356.786,-370
    ]
);
__motion_Wheel.addPropertyArray("blendMode", ["normal"]);
```

En el siguiente ejemplo, el objeto de visualización llamado `Leaf_1` se mueve por el escenario. Sus conjuntos de propiedades `x` e `y` contienen distintos valores en cada uno de los 100 fotogramas de la animación. Además, el objeto gira alrededor de su eje `z` a medida que se mueve por el escenario. Los distintos elementos del conjunto de la propiedad `rotationZ` determinan la rotación.

```
__motion_Leaf_1 = new MotionBase();
__motion_Leaf_1.duration = 100;
__motion_Symbol1_4.addPropertyArray("y",
    [
        0,5.91999,11.84,17.76,23.68,29.6,35.52,41.44,47.36,53.28,59.2,65.12,71.04,
        76.96,82.88,88.8,94.72,100.64,106.56,112.48,118.4,124.32,130.24,136.16,142.08,
        148,150.455,152.909,155.364,157.818,160.273,162.727,165.182,167.636,170.091,
        172.545,175,177.455,179.909,182.364,184.818,187.273,189.727,192.182,194.636,
        197.091,199.545,202,207.433,212.865,218.298,223.73,229.163,234.596,240.028,
        245.461,250.893,256.326,261.759,267.191,272.624,278.057,283.489,
        288.922,294.354,299.787,305.22,310.652,316.085,321.517,326.95,330.475,334,
        337.525,341.05,344.575,348.1,351.625,355.15,358.675,362.2,365.725,369.25,
        372.775,376.3,379.825,383.35,386.875,390.4,393.925,397.45,400.975,404.5,
        407.5,410.5,413.5,416.5,419.5,422.5,425.5
    ]
);
__motion_Symbol1_4.addPropertyArray("scaleX", [1.00]);
__motion_Symbol1_4.addPropertyArray("scaleY", [1.00]);
__motion_Symbol1_4.addPropertyArray("skewX", [0]);
__motion_Symbol1_4.addPropertyArray("skewY", [0]);
__motion_Symbol1_4.addPropertyArray("z",
    [
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ]
);
__motion_Symbol1_4.addPropertyArray("rotationX", [64.0361]);
__motion_Symbol1_4.addPropertyArray("rotationY", [41.9578]);
__motion_Symbol1_4.addPropertyArray("rotationZ",
    [
        -18.0336,-17.5536,-17.0736,-16.5936,-16.1136,-15.6336,-15.1536,-14.6736,
        -14.1936,-13.7136,-13.2336,-12.7536,-12.2736,-11.7936,-11.3136,-10.8336,
        -10.3536,-9.8736,-9.3936,-8.9136,-8.4336,-7.9536,-7.4736,-6.9936,-6.5136,
        -6.0336,-7.21542,-8.39723,-9.57905,-10.7609,-11.9427,-13.1245,-14.3063,
        -15.4881,-16.67,-17.8518,-19.0336,-20.2154,-21.3972,-22.5791,-23.7609,
        -24.9427,-26.1245,-27.3063,-28.4881,-29.67,-30.8518,-32.0336,-31.0771,
        -30.1206,-29.164,-28.2075,-27.251,-26.2945,-25.338,-24.3814,-23.4249,
        -22.4684,-21.5119,-20.5553,-19.5988,-18.6423,-17.6858,-16.7293,-15.7727
        -14.8162,-13.8597,-12.9032,-11.9466,-10.9901,-10.0336,-10.9427,-11.8518,
        -12.7609,-13.67,-14.5791,-15.4881,-16.3972,-17.3063,-18.2154,-19.1245,
        -20.0336,-20.9427,-21.8518,-22.7609,-23.67,-24.5791,-25.4881,-26.3972,
        -27.3063,-28.2154,-29.1245,-30.0336,-28.3193,-26.605,-24.8907,-23.1765,
        -21.4622,-19.7479,-18.0336
    ]
);
__motion_Symbol1_4.addPropertyArray("blendMode", ["normal"]);
```

Añadir filtros

Si el objeto de destino de una interpolación de movimiento contiene filtros, éstos se añaden con los métodos `initFilters()` y `addFilterPropertyArray()` de la clase Motion.

Inicialización del conjunto de filtros

El método `initFilters()` inicializa los filtros. Su primer argumento es un conjunto con los nombres de clases cualificados de todos los filtros aplicados al objeto de visualización. Este conjunto de nombres de filtros se genera a partir de la lista de filtros de la interpolación de movimiento en Flash. En su copia del script puede eliminar o añadir cualquiera de los filtros del paquete `flash.filters` a este conjunto. La siguiente llamada inicializa la lista de filtros del objeto de visualización de destino. Aplica los efectos `DropShadowFilter`, `GlowFilter` y `BevelFilter`. Además, copia la lista en todos los fotogramas clave del objeto Motion.

```
__motion_Box.initFilters(["flash.filters.DropShadowFilter", "flash.filters.GlowFilter",
"flash.filters.BevelFilter"], [0, 0, 0]);
```

Añadir filtros

El método `addFilterPropertyArray()` describe las propiedades de un filtro inicializado con los siguientes argumentos:

1. El primer argumento identifica un filtro por su índice. El índice hace referencia a la posición del nombre del filtro en el conjunto de nombres de la clase Filter transferidos en una llamada previa a `initFilters()`.
2. El segundo argumento es la propiedad del filtro que se guarda para dicho filtro en cada fotograma clave.
3. El tercer argumento es el valor de la propiedad del filtro especificada.

Con una llamada previa a `initFilters()`, las siguientes llamadas a `addFilterPropertyArray()` asignan un valor de 5 a las propiedades `blurX` y `blurY` de `DropShadowFilter`. El elemento `DropShadowFilter` es el primero (índice 0) del conjunto de filtros inicializados:

```
__motion_Box.addFilterPropertyArray(0, "blurX", [5]);
__motion_Box.addFilterPropertyArray(0, "blurY", [5]);
```

Las siguientes tres llamadas asignan valores a las propiedades de calidad, alfa y color de `GlowFilter`, el segundo elemento (índice 1) del conjunto de filtros inicializados:

```
__motion_Box.addFilterPropertyArray(1, "quality", [BitmapFilterQuality.LOW]);
__motion_Box.addFilterPropertyArray(1, "alpha", [1.00]);
__motion_Box.addFilterPropertyArray(1, "color", [0xff0000]);
```

Las siguientes cuatro llamadas asignan valores a las propiedades `shadowAlpha`, `shadowColor`, `highlightAlpha` y `highlightColor` de `BevelFilter`, el tercer elemento (índice 2) del conjunto de filtros inicializados:

```
__motion_Box.addFilterPropertyArray(2, "shadowAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "shadowColor", [0x000000]);
__motion_Box.addFilterPropertyArray(2, "highlightAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "highlightColor", [0xffffff]);
```

Ajuste del color con ColorMatrixFilter

Una vez inicializado el filtro `ColorMatrixFilter`, es posible definir las propiedades adecuadas de `AdjustColor` para ajustar el brillo, el contraste, la saturación y el tono del objeto de visualización interpolado. Normalmente, se aplica el filtro `AdjustColor` en Flash y se perfecciona en la copia del código ActionScript. El siguiente ejemplo transforma el tono y la saturación del objeto de visualización a medida que se mueve.

```
__motion_Leaf_1.initFilters(["flash.filters.ColorMatrix"], [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorBrightness", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorContrast", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorSaturation",
    [
        0,-0.589039,1.17808,-1.76712,-2.35616,-2.9452,-3.53424,-4.12328,
        -4.71232,-5.30136,-5.89041, 6.47945,-7.06849,-7.65753,-8.24657,
        -8.83561,-9.42465,-10.0137,-10.6027,-11.1918,11.7808,-12.3699,
        -12.9589,-13.5479,-14.137,-14.726,-15.3151,-15.9041,-16.4931,
        17.0822,-17.6712,-18.2603,-18.8493,-19.4383,-20.0274,-20.6164,
        -21.2055,-21.7945,22.3836,-22.9726,-23.5616,-24.1507,-24.7397,
        -25.3288,-25.9178,-26.5068,-27.0959,27.6849,-28.274,-28.863,-29.452,
        -30.0411,-30.6301,-31.2192,-31.8082,-32.3973,32.9863,-33.5753,
        -34.1644,-34.7534,-35.3425,-35.9315,-36.5205,-37.1096,-37.6986,
        38.2877,-38.8767,-39.4657,-40.0548,-40.6438,-41.2329,-41.8219,
        -42.411,-43
    ],
    -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorHue",
    [
        0,0.677418,1.35484,2.03226,2.70967,3.38709,4.06451,4.74193,5.41935,
        6.09677,6.77419,7.45161,8.12903,8.80645,9.48387,10.1613,10.8387,11.5161,
        12.1935,12.871,13.5484,14.2258,14.9032,15.5806,16.2581,16.9355,17.6129,
        18.2903,18.9677,19.6452,20.3226,21,22.4286,23.8571,25.2857,26.7143,28.1429,
        29.5714,31,32.4286,33.8571,35.2857,36.7143,38.1429,39.5714,41,42.4286,43.8571,
        45.2857,46.7143,48.1429,49.5714,51,54,57,60,63,66,69,72,75,78,81,84,87,
        90,93,96,99,102,105,108,111,114
    ],
    -1, -1);
```

Asociación de una interpolación de movimiento con sus objetos de visualización

La última tarea es asociar la interpolación de movimiento al objeto u objetos de visualización que manipula.

La clase AnimatorFactory gestiona la asociación entre una interpolación de movimiento y sus objetos de visualización de destino. El argumento del constructor AnimatorFactory es el objeto Motion:

```
var __animFactory_Wheel:AnimatorFactory = new AnimatorFactory(__motion_Wheel);
```

Utilice el método `addTarget()` de la clase AnimatorFactory para asociar el objeto de destino con su interpolación de movimiento. El código ActionScript copiado desde Flash comenta la línea `addTarget()` y no especifica ningún nombre de instancia:

```
// __animFactory_Wheel.addTarget(<instance name goes here>, 0);
```

En su copia, especifique el objeto de visualización que se asociará con la interpolación de movimiento. En el siguiente ejemplo, los destinos se especifican como `greenWheel` y `redWheel`:

```
__animFactory_Wheel.AnimatorFactory.addTarget(greenWheel, 0);
__animFactory_Wheel.AnimationFactory.addTarget(redWheel, 0);
```

Puede asociar varios objetos de visualización con la misma interpolación de movimiento llamando varias veces a `addTarget()`.

Capítulo 20: Trabajo con cinemática inversa

La cinemática inversa (IK) es una gran técnica para crear movimientos realistas.

La IK permite crear movimientos coordinados dentro de una cadena de partes conectadas denominada esqueleto IK, de modo que las partes se mueven juntas como en la vida real. Las partes del esqueleto son sus huesos y sus uniones. Dado el punto final del esqueleto, IK calcula los ángulos necesarios de las uniones para alcanzar dicho punto final.

Calcular estos ángulos manualmente sería todo un reto de programación. Lo atractivo de esta función es la posibilidad de crear esqueletos interactivamente con Adobe® Flash® CS4 Professional. Y, posteriormente, animarlos con ActionScript. La herramienta IK incluida con la herramienta de edición de Flash realiza los cálculos para describir el movimiento del esqueleto. Es posible limitar el movimiento a determinados parámetros del código ActionScript.

Para crear esqueletos de cinemática inversa, debe disponer de una licencia de Adobe Professional (Flash CS4).

Fundamentos de cinemática inversa

Introducción a la cinemática inversa

La cinemática inversa (IK) permite crear animaciones reales uniendo distintas piezas de modo que puedan moverse entre sí con realismo.

Por ejemplo, se puede utilizar la cinemática inversa para mover una pierna hasta una posición determinada articulando los movimientos necesarios de las uniones de la pierna para lograr la postura deseada. La cinemática inversa utiliza una arquitectura de huesos unidos entre sí en una estructura denominada esqueleto IK. El paquete `fl.ik` sirve para crear animaciones de movimientos naturales y realistas. Permite animar varios esqueletos IK a la perfección sin necesidad de tener conocimientos físicos de algoritmos de cinemática inversa.

Puede crear el esqueleto IK con sus huesos y uniones correspondientes en Flash. Luego puede acceder a las clases de IK para animarlos en tiempo de ejecución.

Consulte la sección Utilización de cinemática inversa en el manual *Utilización de Flash CS4 Professional* para obtener instrucciones detalladas sobre la creación de un esqueleto IK.

Tareas comunes de cinemática inversa

El código ActionScript para iniciar y controlar el movimiento de un esqueleto IK en tiempo de ejecución suele hacer lo siguiente:

- Declarar variables para los esqueletos, huesos y uniones implicados en el movimiento
- Recuperar las instancias del esqueleto, huesos y uniones
- Crear instancias del objeto mover de IK
- Definir límites en el movimiento
- Mover el esqueleto hasta un punto de destino

Términos y conceptos importantes

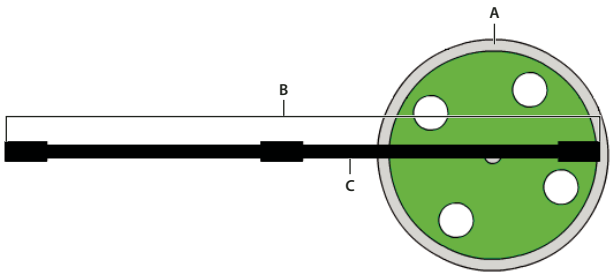
La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Esqueleto: cadena cinemática formada por huesos y uniones que se utiliza en animación por ordenador para simular movimientos con realismo
- Hueso: segmento rígido de un esqueleto (análogo al hueso de un esqueleto animal)
- Cinemática inversa (IK): proceso por el cual se determinan los parámetros de un objeto flexible unido denominado cadena cinemática o esqueleto
- Unión: lugar en el que dos huesos entran en contacto, creado para permitir el movimiento de los huesos; es similar a la articulación de un animal.

Información general sobre animación de esqueletos IK

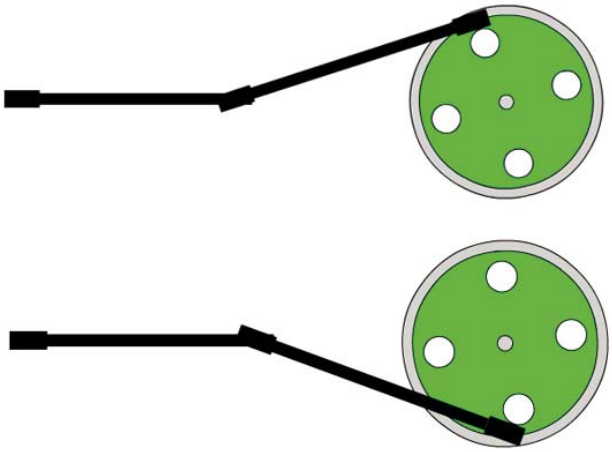
Una vez creado el esqueleto IK, utilice las clases de `fl.ik` para limitar su movimiento, realizar un seguimiento de sus eventos o animarlo en tiempo de ejecución.

En la siguiente figura se muestra un clip de película denominado `Rueda`. El eje es una instancia de un objeto IKArmature denominado `Eje`. La clase IKMover mueve el esqueleto en sincronía con la rotación de la rueda. El hueso IKBone (`ikBone2`) del esqueleto se une a la rueda en su unión de cola.



***A.** Rueda **B.** Eje **C.** ikBone2*

En tiempo de ejecución, la rueda gira asociada a la interpolación de movimiento `__motion_Wheel` descrita en "Descripción de la animación" en la página 433 en el capítulo Trabajo con interpolaciones de movimiento. Un objeto IKMover inicia y controla el movimiento del eje. En la siguiente figura se muestran dos instantáneas del esqueleto del eje asociado a la rueda giratoria en distintos fotogramas de la rotación.



En tiempo de ejecución, el siguiente código ActionScript:

- Obtiene información sobre el esqueleto y sus componentes
- Crea una instancia del objeto IKMover
- Mueve el eje junto con la rotación de la rueda

```
import fl.ik.*

var tree:IKArmature = IKManager.getArmatureByName("Axle");
var bone:IKBone = tree.getBoneByName("ikBone2");
var endEffector:IKJoint = bone.tailJoint;
var pos:Point = endEffector.position;

var ik:IKMover = new IKMover(endEffector, pos);
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;

Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90, 0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

Las clases IK utilizadas para mover el eje son:

- IKArmature: describe el esqueleto, una estructura en árbol formada por huesos y uniones. Se debe crear con Flash
- IKManager: clase contenedora de todos los esqueletos IK del documento. Se debe crear con Flash
- IKBone: segmento de un esqueleto IK
- IKJoint: conexión entre dos huesos IK
- IKMover: inicia y controla el movimiento IK de los esqueletos

Para obtener información detallada y completa sobre estas clases, consulte el paquete ik.

Obtener información sobre un esqueleto IK

En primer lugar, declare variables para el esqueleto, el hueso y la unión que formen parte del grupo de piezas que desea mover.

El siguiente código utiliza el método `getArmatureByName()` de la clase IKManager para asignar el valor del esqueleto Axle a la variable `tree` de IKArmature. El esqueleto Axle se ha creado previamente con Flash.

```
var tree:IKArmature = IKManager.getArmatureByName("Axle");
```

De forma similar, el siguiente código utiliza el método `getBoneByName()` de la clase IKArmature para asignar a la variable IKBone el valor del hueso `ikBone2`.

```
var bone:IKBone = tree.getBoneByName("ikBone2");
```

La unión de cola del hueso `ikBone2` es la parte del esqueleto que se une a la rueda giratoria.

La siguiente línea declara la variable `endEffector` y le asigna la propiedad `tailjoint` del hueso `ikBone2`:

```
var endEffector:IKJoint = home.tailjoint;
```

La variable `pos` es un punto que almacena la posición actual de la unión `endEffector`.

```
var pos:Point = endEffector.position;
```

En este ejemplo, `pos` corresponde a la posición de la unión al final del eje, donde se conecta con la rueda. El valor original de esta variable se obtiene a partir de la propiedad `position` del objeto IKJoint.

Creación de una instancia de la clase IKMover y limitación del movimiento

Una instancia de la clase IKMover mueve el eje.

La siguiente línea crea una instancia del objeto `ik` de IKMover, transfiere a su constructor el elemento que desea mover y el punto de inicio del movimiento:

```
var ik:IKMover = new IKMover(endEffector, pos);
```

Las propiedades de la clase IKMover permiten limitar el movimiento de un esqueleto. Puede limitar el movimiento basado en la distancia, en las iteraciones o en el tiempo del movimiento.

Las siguientes parejas de propiedades refuerzan estos límites. La pareja consta de una propiedad `Boolean` que indica si el movimiento está limitado y el número que indica dicho límite:

`limitByDistance:Boolean`	`distanceLimit:int`	Establece la distancia máxima (en píxeles) que se mueve el motor IK en cada iteración.
`limitByIteration:Boolean`	`iterationLimit:int`	Especifica el número máximo de iteraciones que realiza el motor IK en cada movimiento.
`limitByTime:Boolean`	`timeLimit:int`	Establece el tiempo máximo (en milisegundos) asignado al motor IK para realizar el movimiento.

Establezca la propiedad `Boolean` correspondiente como `true` para reforzar el límite. De forma predeterminada, todas las propiedades `Boolean` se establecen como `false`, por lo que el movimiento no está limitado a no ser que se establezcan explícitamente. Si establece el límite en un valor sin definir su propiedad `Boolean` correspondiente, el límite se obviará. En este caso, el motor IK sigue moviendo el objeto hasta llegar a otro límite o a la posición de destino del objeto IKMover.

En el siguiente ejemplo, la distancia máxima del movimiento del esqueleto se establece en 0,1 píxeles por iteración. El número máximo de iteraciones de cada movimiento se establece en diez.

```
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;
```

Desplazamiento de un esqueleto IK

El objeto IKMover mueve el eje dentro del detector de eventos de la rueda. En cada evento enterFrame de la rueda, se calcula una nueva posición de destino para el esqueleto. Si se utiliza su método `moveTo()`, el objeto IKMover mueve la unión de cola hasta su posición de destino o tan lejos como lo permitan las limitaciones definidas por las propiedades `limitByDistance`, `limitByIteration` y `limitByTime`.

```
Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90,0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

Utilización de eventos IK

La clase IKEvent permite crear un objeto de evento que contenga información sobre eventos IK. La información de IKEvent describe un movimiento que ha finalizado por haberse superado el tiempo especificado, la distancia o el límite de iteraciones.

El siguiente código muestra un detector de eventos y un controlador para realizar el seguimiento de los eventos de límite de tiempo. Este controlador de eventos informa sobre el tiempo, la distancia, el número de iteraciones y las propiedades de unión de un evento activado cuando se supera el límite de tiempo de IKMover.

```
var ikmover:IKMover = new IKMover(endjoint, pos);
ikMover.limitByTime = true;
ikMover.timeLimit = 1000;

ikmover.addEventListener(IKEvent.TIME_LIMIT, timeLimitFunction);

function timeLimitFunction(evt:IKEvent):void
{
    trace("timeLimit hit");
    trace("time is " + evt.time);
    trace("distance is " + evt.distance);
    trace("iterationCount is " + evt.iterationCount);
    trace("IKJoint is " + evt.joint.name);
}
```

Capítulo 21: Trabajo con texto

Para mostrar texto en pantalla en Adobe® Flash® Player o Adobe® AIR™, se utiliza una instancia de la clase TextField o las clases de Flash Text Engine. Estas clases permiten crear texto, presentarlo y aplicarle formato.

Se puede establecer contenido específico para los campos de texto, o designar el origen del texto, y a continuación definir el aspecto de dicho texto. También se puede responder a eventos de usuario, como cuando el usuario introduce texto o hace clic en un vínculo de hipertexto.

Fundamentos de la utilización de texto

Introducción a la utilización de texto

Tanto la clase TextField como la clases de Flash Text Engine permiten mostrar y administrar texto en Flash Player y AIR.

La clase TextField se puede utilizar para crear objetos de texto para visualización y entrada. TextField proporciona la base para otros componentes basados en texto, como TextArea y TextInput incluidos en Flash y Adobe Flex. La clase TextFormat se puede utilizar para establecer el formato de párrafo y caracteres para los objetos TextField y se pueden aplicar hojas de estilos en cascada (CSS) con el uso de la propiedad Textfield.styleSheet y de la clase StyleSheet. Se puede asignar un texto con formato HTML, que puede contener elementos multimedia incorporados (clips de película y archivos SWF, GIF, PNG y JPEG), directamente a un campo de texto.

Flash Text Engine, disponible a partir de Flash Player 10 y Adobe AIR 1.5, ofrece compatibilidad de bajo nivel para realizar un sofisticado control de las medidas y el formato del texto, además de admitir texto bidireccional. También proporciona una compatibilidad para idiomas y un flujo de texto mejorados. Aunque Flash Text Engine se puede utilizar para crear y administrar elementos de texto, principalmente está diseñado como base para la creación de componentes de gestión del texto y requiere una experta labor de tareas de programación.

Tareas comunes para trabajar con texto

Las siguientes tareas comunes se llevan a cabo con la clase TextField:

- Modificación del contenido de un campo de texto
- Utilización de HTML en campos de texto
- Utilización de imágenes en campos de texto
- Selección de texto y trabajo con texto seleccionado por el usuario
- Captura de la entrada de texto
- Restricción de la entrada de texto
- Aplicación de formato y estilos CSS a texto
- Ajuste fino de la visualización del texto ajustando la nitidez, el grosor y el suavizado
- Acceso a campos de texto estáticos desde ActionScript y utilizarlos

Las siguientes tareas comunes se llevan a cabo con las clases de Flash Text Engine:

- Creación y visualización de texto
- Gestión de eventos

- Formato de texto
- Trabajo con fuentes
- Control del texto

Conceptos y términos importantes

La siguiente lista de referencia contienen los términos importantes que se utilizan:

- Hojas de estilos en cascada: sintaxis estándar para especificar estilos y formato para contenido estructurado con formato XML (o HTML).
- Fuente de dispositivo: fuente instalada en el equipo del usuario.
- Campo de texto dinámico: campo de texto cuyo contenido se puede modificar mediante código ActionScript pero no con una entrada de usuario.
- Fuente incorporada: fuente que tiene los datos de contorno de caracteres almacenados en el archivo SWF de la aplicación.
- Texto HTML: contenido de texto introducido en un campo de texto mediante código ActionScript que incluye etiquetas de formato HTML junto con el contenido de texto.
- Campo de texto de entrada: campo de texto cuyo contenido se puede modificar mediante una entrada de usuario o mediante código ActionScript.
- Espaciado manual: ajuste del espaciado entre pares de caracteres para que el espaciado de las palabras sea más proporcional y se facilite la lectura del texto.
- Campo de texto estático: campo de texto creado en la herramienta de edición, cuyo contenido no se puede modificar mientras se ejecuta el archivo SWF.
- Medidas de líneas de texto: medidas de las diversas partes del contenido de texto de un campo de texto, como la línea de base del texto, la altura de la parte superior de los caracteres, el tamaño de los trazos descendentes (la parte de algunas letras minúsculas que se extiende por debajo de la línea de base), etc.
- Espaciado entre caracteres: ajuste del espaciado entre grupos de letras o bloques de texto para aumentar o disminuir la densidad y hacer el texto más legible.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Prácticamente todos los listados de código implican manipular un objeto text, ya se haya creado y colocado en el escenario mediante la herramienta de edición de Flash, o utilizando código ActionScript. Para probar el ejemplo hay que ver el resultado en Flash Player o AIR, con el fin de ver los efectos del código en el texto o el objeto TextField.

Los ejemplos de este tema se dividen en dos grupos. Hay un tipo de ejemplo que manipula un objeto TextField sin crear el objeto explícitamente. Para probar estos listados de código, siga estos pasos:

1. Cree un documento de Flash vacío.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Con la herramienta Texto, cree un campo de texto dinámico en el escenario.

5 Con el campo de texto seleccionado, asígnele un nombre de instancia en el inspector de propiedades. El nombre debe coincidir con el utilizado para el campo de texto en el listado de código de ejemplo; por ejemplo, si el código manipula un campo de texto denominado `myTextField`, debe asignar a la instancia de campo de texto el nombre `myTextField`.

6 Ejecute el programa seleccionando Control > Probar película.

Verá en pantalla el resultado de la manipulación del campo de texto realizada por el código.

El otro tipo de listado de código de ejemplo consta de una definición de clase que se utilizará como la clase de documento para el archivo SWF. En estos listados, los objetos de texto se crean con código de ejemplo, por lo que no es necesario crearlos por separado. Para probar este tipo de listado de código:

1 Cree un documento de Flash vacío y guárdelo en el equipo.

2 Cree un nuevo archivo de ActionScript y guárdelo en el mismo directorio que el documento de Flash. El nombre del archivo debe coincidir con el nombre de la clase del listado de código. Por ejemplo, si el listado de código define una clase denominada TextFieldTest, use el nombre TextFieldTest.as para guardar el archivo de ActionScript.

3 Copie el listado de código en el archivo de ActionScript y guarde el archivo.

4 En el documento de Flash, seleccione Ventana -> Propiedades para activar el inspector de propiedades del documento.

5 En el inspector de propiedades, en el campo Clase de documento, escriba el nombre de la clase de ActionScript que copió del texto.

6 Ejecute el programa seleccionando Control > Probar película.

Observe el resultado del ejemplo mostrado en pantalla.

Ésta y otras técnicas para probar los listados de código de ejemplo se describen de forma detallada en “Prueba de los listados de código de ejemplo del capítulo” en la página 36.

Utilización de la clase TextField

La clase TextField es la base de otros componentes basados texto, como TextArea o los componentes TextInput, que se proporcionan en la arquitectura Adobe Flex y en el entorno de edición de Flash. Para más información sobre cómo utilizar componentes de texto en el entorno de edición de Flash, consulte "Controles de texto" en *Utilización de Flash*.

El contenido de un campo de texto se puede preespecificar en el archivo SWF, se puede cargar desde un archivo de texto o una base de datos, o puede ser introducido por un usuario que interactúe con la aplicación. En un campo de texto, el texto puede aparecer como contenido HTML representado con imágenes incorporadas. Una vez creada una instancia de un campo de texto, se pueden utilizar las clases flash.text, como TextFormat y StyleSheet, para controlar la apariencia del texto. El paquete flash.text package contiene casi todas las clases relacionadas con la creación, administración y formato de texto en ActionScript.

Se puede aplicar formato a texto definiendo el formato con un objeto TextFormat y asignando dicho objeto al campo de texto. Si el campo de texto contiene texto HTML, se puede aplicar un objeto StyleSheet al campo de texto para asignar estilos a partes específicas del contenido del campo de texto. El objeto TextFormat o el objeto StyleSheet contienen propiedades que definen la apariencia del texto, como el color, el tamaño y el grosor. El objeto TextFormat asigna las propiedades a todo el contenido de un campo de texto o a un rango de texto. Por ejemplo, en un mismo campo de texto una frase puede estar en negrita y de color rojo y la frase siguiente en cursiva y de color azul.

Para más información sobre los formatos de texto, consulte “Asignación de formatos de texto” en la página 453.

Para más información sobre texto HTML en campos de texto, consulte"Visualización de texto HTML" en la página 448.

Para más información sobre las hojas de estilo, consulte "Aplicación de hojas de estilos en cascada" en la página 454.

Además de las clases del paquete flash.text, se puede utilizar la clase flash.events.TextEvent para responder a acciones del usuario relacionadas con texto.

Visualización de texto

Aunque las herramientas de edición como Adobe Flex Builder y la herramienta de edición de Flash proporcionan varias opciones para visualizar texto, como componentes relacionados con texto o herramientas de texto, la principal manera de mostrar texto mediante programación es utilizar un campo de texto.

Tipos de texto

El tipo de texto de un campo de texto se caracteriza por su origen:

- Texto dinámico

  El texto dinámico incluye contenido que se carga desde un origen externo, como un archivo de texto, un archivo XML o incluso un servicio Web remoto.

- Texto de entrada

  El texto de entrada es cualquier texto introducido por un usuario o texto dinámico que un usuario puede editar. Se puede establecer una hoja de estilos para aplicar formato al texto de entrada o utilizar la clase flash.text.TextFormat para asignar propiedades al campo de texto para el contenido de la entrada. Para más información, consulte "Captura de una entrada de texto" en la página 451.

- Texto estático

  El texto estático sólo se crea en la herramienta de edición Con ActionScript 3.0 no se pueden crear instancias de texto estático. Sin embargo, se pueden usar clases de ActionScript como StaticText y TextSnapshot para manipular instancias de texto estático existentes. Para más información, consulte "Trabajo con texto estático" en la página 458.

Edición del contenido de un campo de texto

Puede definir texto dinámico asignando una cadena a la propiedad `flash.text.TextField.text`. Se puede asignar una cadena directamente a la propiedad, de la manera siguiente:

```
myTextField.text = "Hello World";
```

También se puede asignar a la propiedad `text` un valor de una variable definida en el script, como en el siguiente ejemplo:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello World";

        public function TextWithImage()
        {
            addChild(myTextBox);
            myTextBox.text = myText;
        }
    }
}
```

Como alternativa, se puede asignar a la propiedad `text` un valor de una variable remota. Hay tres opciones para cargar valores de texto desde orígenes remotos:

- Las clases flash.net.URLLoader y flash.net.URLRequest cargan variables para el texto desde una ubicación local o remota.
- El atributo `FlashVars` está incorporado en la página HTML que aloja el archivo SWF y puede contener valores para variables de texto.
- La clase flash.net.SharedObject administra el almacenamiento persistente de valores. Para más información, consulte "Almacenamiento de datos locales" en la página 638.

Visualización de texto HTML

La clase flash.text.TextField tiene una propiedad `htmlText` que se puede utilizar para identificar la cadena de texto como una cadena que contiene etiquetas HTML para aplicar formato al contenido. Al igual que en el siguiente ejemplo, se debe asignar el valor de cadena a la propiedad `htmlText` (no a la propiedad `text`) para que Flash Player o AIR represente el texto en formato HTML:

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Flash Player y AIR admiten un subconjunto de etiquetas y entidades HTML para la propiedad `htmlText`. La descripción de la propiedad `flash.text.TextField.htmlText` en la Referencia del lenguaje y componentes ActionScript 3.0 proporciona información detallada sobre las etiquetas y entidades HTML admitidas.

Una vez designado el contenido mediante la propiedad `htmlText`, se pueden utilizar hojas de estilos o la etiqueta `textformat` para administrar el formato del contenido. Para más información, consulte "Formato de texto" en la página 453.

Utilizar imágenes en campos de texto

Otra ventaja de mostrar el contenido como texto HTML es que se pueden incluir imágenes en el campo de texto. Se puede hacer referencia a una imagen, local o remota, mediante la etiqueta `img` y hacer que aparezca dentro del campo de texto asociado.

En el ejemplo siguiente se crea un campo de texto denominado `myTextBox` y se incluye una imagen JPG de un ojo, almacenada en el mismo directorio que el archivo SWF, dentro del texto mostrado:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField;
        private var myText:String = "<p>This is <b>some</b> content to <i>test</i> and
<i>see</i></p><p><img src='eye.jpg' width='20' height='20'></p><p>what can be
rendered.</p><p>You should see an eye image and some <u>HTML</u> text.</p>";

        public function TextWithImage()
        {
            myTextBox.width = 200;
            myTextBox.height = 200;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.border = true;

            addChild(myTextBox);
            myTextBox.htmlText = myText;
        }
    }
}
```

La etiqueta `img` admite archivos JPEG, GIF, PNG y SWF.

Desplazamiento de texto en un campo de texto

En muchos casos, el texto puede ser más largo que el campo de texto que muestra el texto. O se puede tener un campo de entrada que permite a un usuario introducir más texto que el que se puede mostrar de una sola vez. Se pueden utilizar las propiedades relacionadas con el desplazamiento de la clase flash.text.TextField para administrar contenido extenso, tanto verticalmente como horizontalmente.

Las propiedades relacionadas con el desplazamiento incluyen `TextField.scrollV`, `TextField.scrollH`, `maxScrollV` y `maxScrollH`. Estas propiedades pueden utilizarse para responder a eventos, como un clic del ratón o una pulsación de tecla.

En el ejemplo siguiente se crea un campo de texto con un tamaño establecido y que contiene más texto que el campo puede mostrar de una sola vez. A medida que el usuario hace clic en el campo de texto, el texto se desplaza verticalmente.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;
    import flash.events.MouseEvent;

    public class TextScrollExample extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello world and welcome to the show. It's really nice to
meet you. Take your coat off and stay a while. OK, show is over. Hope you had fun. You can go
home now. Don't forget to tip your waiter. There are mints in the bowl by the door. Thank you.
Please come again.";

        public function TextScrollExample()
        {
            myTextBox.text = myText;
            myTextBox.width = 200;
            myTextBox.height = 50;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.background = true;
            myTextBox.border = true;

            var format:TextFormat = new TextFormat();
            format.font = "Verdana";
            format.color = 0xFF0000;
            format.size = 10;

            myTextBox.defaultTextFormat = format;
            addChild(myTextBox);
            myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
        }

        public function mouseDownScroll(event:MouseEvent):void
        {
            myTextBox.scrollV++;
        }
    }
}
```

Selección y manipulación de texto

Se puede seleccionar texto dinámico o de entrada. Como las propiedades y los métodos de selección de texto de la clase TextField utilizan posiciones de índice para establecer el rango de texto que se va a manipular, se puede seleccionar mediante programación texto dinámico o de entrada aunque no se conozca su contenido.

Nota: *en la herramienta de edición de Flash, si se elige la opción seleccionable en un campo de texto estático, el campo de texto que se exporta y coloca en la lista de visualización es un campo de texto dinámico normal.*

Selección de texto

El valor de la propiedad `flash.text.TextField.selectable` es `true` de manera predeterminada, y se puede seleccionar texto mediante programación a través del método `setSelection()`.

Por ejemplo, se puede establecer texto específico de un campo de texto que debe ser seleccionado cuando el usuario haga clic en el campo de texto:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
    myTextField.setSelection(49, 65);
}
```

De forma similar, si se desea que se seleccione texto de un campo de texto como el texto que se mostrará inicialmente, se debe crear una función de controlador de eventos a la que se llama cuando se añade texto a la lista de visualización.

Captura de texto seleccionado por el usuario

Las propiedades `selectionBeginIndex` y `selectionEndIndex` de TextField, que son de "sólo lectura" para que no se pueden establecer sus valores mediante programación, pueden utilizarse para capturar lo que el usuario haya seleccionado actualmente. Además, los campos de entrada de texto pueden utilizar la propiedad `caretIndex`.

Por ejemplo, el código siguiente hace un seguimiento de lo valores de índice de texto seleccionado por el usuario:

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index values for the first
and last letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
    trace("First letter index position: " + myTextField.selectionBeginIndex);
    trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

Se puede aplicar una colección de propiedades de objeto TextFormat a la selección para cambiar el aspecto del texto. Para más información sobre la aplicación de una colección de propiedades de TextFormat a texto seleccionado, consulte "Formato de rangos de texto en un campo de texto" en la página 456.

Captura de una entrada de texto

De manera predeterminada, la propiedad `type` de un campo de texto se establece en `dynamic`. Si se establece la propiedad `type` en `input` mediante la clase TextFieldType, se puede obtener la entrada del usuario y guardar el valor para utilizarlo en otras partes de la aplicación. Los campos de entrada de texto son útiles para los formularios y para cualquier aplicación que requiera que el usuario defina un valor de texto para utilizarlo en otro lugar del programa.

Por ejemplo, el código siguiente crea un campo de entrada de texto llamado `myTextBox`. Cuando el usuario escribe texto en el campo, se activa el evento `textInput`. Un controlador de eventos denominado `textInputCapture` captura la cadena de texto introducido y se la asigna a una variable. Flash Player o AIR muestran el nuevo texto en otro campo de texto, denominado `myOutputBox`.

```
package
{
    import flash.display.Sprite;
    import flash.display.Stage;
    import flash.text.*;
    import flash.events.*;

    public class CaptureUserInput extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myOutputBox:TextField = new TextField();
        private var myText:String = "Type your text here.";

        public function CaptureUserInput()
        {
            captureText();
        }

        public function captureText():void
        {
            myTextBox.type = TextFieldType.INPUT;
            myTextBox.background = true;
            addChild(myTextBox);
            myTextBox.text = myText;
            myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
        }

        public function textInputCapture(event:TextEvent):void
        {
            var str:String = myTextBox.text;
            createOutputBox(str);
        }

        public function createOutputBox(str:String):void
        {
            myOutputBox.background = true;
            myOutputBox.x = 200;
            addChild(myOutputBox);
            myOutputBox.text = str;
        }

    }
}
```

Restricción de la entrada de texto

Como los campos de entrada de texto se suelen utilizar para formularios o cuadros de diálogo en aplicaciones, es posible que se desee limitar los tipos de caracteres que un usuario puede introducir en un campo de texto o incluso mantener el texto oculto (por ejemplo, para una contraseña). La clase flash.text.TextField tiene una propiedad `displayAsPassword` y una propiedad`restrict` que se pueden establecer para controlar la entrada del usuario.

La propiedad `displayAsPassword` simplemente oculta el texto (mostrándolo como una serie de asteriscos) que escribe el usuario. Cuando `displayAsPassword` está establecida en `true`, los comandos Cortar y Copiar y sus correspondientes métodos abreviados de teclado no funcionan. Como se muestra en el siguiente ejemplo, se asigna la propiedad `displayAsPassword` de la misma manera que otras propiedades, como background y color:

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

La propiedad `restrict` es un poco más complicada, ya que hay que especificar qué caracteres puede escribir el usuario en un campo de entrada de texto. Se pueden permitir letras, números o rangos de letras, números y caracteres específicos. El código siguiente permite al usuario escribir únicamente letras mayúsculas (y no números ni caracteres especiales) en el campo de texto:

```
myTextBox.restrict = "A-Z";
```

ActionScript 3.0 utiliza guiones para definir rangos y caracteres ^ para definir caracteres excluidos. Para más información sobre cómo definir restricciones en un campo de entrada de texto, consulte la entrada sobre la propiedad `flash.text.TextField.restrict` en la Referencia del lenguaje y componentes ActionScript 3.0.

Formato de texto

Hay varias opciones para aplicar formato a la visualización del texto mediante programación. Se pueden establecer propiedades directamente en la instancia de TextField (por ejemplo, las propiedades `TextFIeld.thickness`, `TextField.textColor` y `TextField.textHeight`.O se puede designar el contenido del campo de texto mediante la propiedad `htmlText` y utilizar las etiquetas HTML admitidas, como `b`, `i` y `u`. Pero también se pueden aplicar objetos TextFormat a campos de texto que contienen texto simple u objetos StyleSheet a campos de texto que contienen la propiedad `htmlText`. La utilización de objetos TextFormat y StyleSheet proporciona el mayor control y la mayor uniformidad de la apariencia del texto en toda la aplicación. Se puede definir un objeto TextFormat o StyleSheet, y aplicárselo a muchos de los campos de texto de la aplicación (o a todos ellos).

Asignación de formatos de texto

Se puede utilizar la clase TextFormat para establecer varias propiedades de visualización de texto distintas y para aplicarlas a todo el contenido de un objeto TextField o a un rango de texto.

En el ejemplo siguiente se aplica un objeto TextFormat a un objeto TextField completo y se aplica un segundo objeto TextFormat a un rango de texto dentro de ese objeto TextField:

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

El método `TextField.setTextFormat()` sólo afecta al texto que ya está visualizado en el campo de texto. Si cambia el contenido del objeto TextField, la aplicación tendrá que volver a llamar al método `TextField.setTextFormat()` para aplicar de nuevo el formato. También se puede establecer la propiedad `defaultTextFormat` de TextField para especificar el formato que se debe utilizar para el texto introducido por el usuario.

Aplicación de hojas de estilos en cascada

Los campos de texto pueden contener texto simple o texto en formato HTML. El texto simple se almacena en la propiedad `text` de la instancia y el texto HTML se almacena en la propiedad `htmlText`.

Se puede utilizar declaraciones de estilos CSS para definir estilos de texto que se pueden aplicar a muchos campos de texto distintos. Las declaraciones de estilos CSS pueden crearse en el código de la aplicación o cargarse en tiempo de ejecución desde un archivo CSS externo.

La clase flash.text.StyleSheet gestiona los estilos CSS. La clase StyleSheet reconoce un conjunto limitado de propiedades CSS. Para ver una lista detallada de propiedades de estilos admitidas por la clase StyleSheet, consulte la entrada de flash.textStylesheet en la Referencia del lenguaje y componentes ActionScript 3.0.

Como se indica en el siguiente ejemplo, se pueden crear hojas de estilos CSS en el código y aplicar dichos estilos a texto HTML mediante un objeto StyleSheet:

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

Tras crear un objeto StyleSheet, el código de ejemplo crea un objeto simple para almacenar un conjunto de propiedades de declaración de estilos. A continuación, llama al método `StyleSheet.setStyle()`, que añade el nuevo estilo a la hoja de estilos con el nombre ".darkred". Por último, aplica el formato de la hoja de estilos asignando el objeto StyleSheet a la propiedad `styleSheet` del objeto TextField.

Para que los estilos CSS surtan efecto, la hoja de estilos debe aplicarse al objeto TextField antes de establecer la propiedad `htmlText`.

Por diseño, un campo de texto con una hoja de estilos no es editable. Si se tiene un campo de entrada de texto y se le asigna una hoja de estilos, el campo de texto muestra las propiedades de la hoja de estilos, pero el campo de texto no permitirá a los usuarios escribir texto nuevo en él. Además, no se pueden utilizar los siguientes métodos de ActionScript en un campo de texto con una hoja de estilos asignada:

- El método `TextField.replaceText()`
- El método `TextField.replaceSelectedText()`
- La propiedad `TextField.defaultTextFormat`
- El método `TextField.setTextFormat()`

Si se ha asignado una hoja de estilos a un campo de texto, pero después se establece la propiedad `TextField.styleSheet` en `null`, el contenido de las propiedades `TextField.text` y `TextField.htmlText` añade etiquetas y atributos a su contenido para incorporar el formato de la hoja de estilos asignada anteriormente. Para conservar la propiedad `htmlText` original, debe guardarse en una variable antes de establecer la hoja de estilos en `null`.

Carga de un archivo CSS externo

El enfoque de utilizar CSS para el formato es más eficaz si se carga información CSS desde un archivo externo en tiempo de ejecución. Cuando los datos CSS son externos a la misma aplicación, se puede cambiar el estilo visual de texto en la aplicación sin tener que modificar el código fuente de ActionScript 3.0. Una vez desplegada la aplicación se puede cambiar un archivo CSS externo para cambiar el aspecto de la aplicación sin tener que volver a desplegar el archivo SWF de la misma.

El método `StyleSheet.parseCSS()` convierte una cadena que contiene datos CSS en declaraciones de estilo del objeto StyleSheet. En el siguiente ejemplo se muestra la manera de leer un archivo CSS externo y se aplican sus declaraciones de estilo a un objeto TextField.

En primer lugar, éste es el contenido del archivo CSS que se va a cargar, denominado example.css:

```
p {
    font-family: Times New Roman, Times, _serif;
    font-size: 14;
}

h1 {
    font-family: Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
}

.bluetext {
    color: #0000CC;
}
```

A continuación se muestra el código ActionScript de una clase que carga el archivo example.css y aplica los estilos al contenido de TextField:

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.text.StyleSheet;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class CSSFormattingExample extends Sprite
    {
        var loader:URLLoader;
        var field:TextField;
        var exampleText:String = "<h1>This is a headline</h1>" +
            "<p>This is a line of text. <span class='bluetext'>" +
            "This line of text is colored blue.</span></p>";

        public function CSSFormattingExample():void
        {
            field = new TextField();
            field.width = 300;
```

```
            field.autoSize = TextFieldAutoSize.LEFT;
            field.wordWrap = true;
            addChild(field);

            var req:URLRequest = new URLRequest("example.css");

            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            var sheet:StyleSheet = new StyleSheet();
            sheet.parseCSS(loader.data);
            field.styleSheet = sheet;
            field.htmlText = exampleText;
        }
    }
}
```

Una vez cargados los datos CSS, se ejecuta el método `onCSSFileLoaded()` y se llama al método `StyleSheet.parseCSS()` para transferir las declaraciones de estilo al objeto StyleSheet.

Formato de rangos de texto en un campo de texto

Un método especialmente útil de la clase flash.text.TextField es `setTextFormat()`. Con `setTextFormat()` se pueden asignar propiedades específicas al contenido de una parte de un campo de texto para responder a una entrada de usuario, como en el caso de los formularios que deben recordar a los usuarios que algunas entradas son obligatorias o para resaltar una parte de un texto dentro de un campo de texto a medida que el usuario selecciona partes del texto.

En el ejemplo siguiente se utiliza `TextField.setTextFormat()` en un rango de caracteres para cambiar el aspecto de una parte del contenido de `myTextField` cuando el usuario hace clic en el campo de texto:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS changes
format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
    myTextField.setTextFormat(myformat, 49, 65);
}
```

Representación de texto avanzada

ActionScript 3.0 proporciona diversas clases en el paquete flash.text para controlar las propiedades del texto mostrado, incluidas las fuentes incorporadas, la configuración de suavizado, el control del canal alfa y otras configuraciones específicas. La Referencia del lenguaje y componentes ActionScript 3.0 proporciona descripciones detalladas de estas clases y propiedades, incluidas las clases CSMSettings, Font y TextRenderer.

Utilización de fuentes incorporadas

Si se especifica una fuente específica para un objeto TextField en la aplicación, Flash Player o AIR buscarán una fuente de dispositivo (una fuente que resida en el equipo del usuario) que tenga el mismo nombre. Si no encuentra esa fuente en el sistema o si el usuario tiene una versión ligeramente distinta de una fuente con ese nombre, el aspecto del texto visualizado puede ser muy distinto del esperado.

Para asegurarse de que el usuario ve la fuente correcta, se puede incorporar dicha fuente en el archivo SWF de la aplicación. Las fuentes incorporadas ofrecen varias ventajas:

- Los caracteres de las fuentes incorporadas se suavizan, por lo que sus bordes parecen más lisos, especialmente en textos grandes.
- Se puede girar textos con fuentes incorporadas.
- El texto de una fuente incorporada se puede convertir en transparente o semitransparente.
- Se puede utilizar el estilo CSS de `kerning (ajuste entre caracteres)` con fuentes incorporadas.

La mayor limitación del uso de fuentes incorporadas es que aumentan el tamaño del archivo o el tiempo de descarga de la aplicación.

El método preciso de incorporar un archivo de fuente en el archivo SWF de la aplicación varía de un entorno de desarrollo a otro.

Tras incorporar una fuente, hay que asegurarse de que un objeto TextField utiliza la fuente incorporada correcta:

- Establezca el valor de la propiedad `embedFonts` del objeto TextField en `true`.
- Cree un objeto TextFormat, establezca su propiedad `fontFamily` en el nombre de la fuente incorporada y aplique el objeto TextFormat al objeto TextField. Al especificar una fuente incorporada, la propiedad `fontFamily` sólo debe contener un único nombre; no se puede utilizar una lista de nombres de fuentes delimitados por comas.
- Si se utilizan estilos CSS para establecer fuentes de objetos TextField o componentes, hay que establecer la propiedad CSS `font-family` en el nombre de la fuente incorporada. Si se desea especificar una fuente incorporada, la propiedad `font-family` debe contener un solo nombre, no una lista de nombres.

Incorporación de una fuente en Flash

La herramienta de edición de Flash permite incorporar prácticamente cualquier fuente que esté instalada en el sistema, incluidas las fuentes TrueType y las fuentes Postscript de tipo 1.

Hay muchas maneras de incorporar fuentes en una aplicación, como:

- Establecer las propiedades de fuente y estilo de un objeto TextField en el escenario y hacer clic en la casilla de verificación Fuentes incorporadas
- Crear un símbolo de fuente y hacer referencia al mismo
- Crear y utilizar una biblioteca compartida en tiempo de ejecución que contenga símbolos de fuente incorporados

Para más detalles sobre cómo incorporar fuentes en aplicaciones , consulte "Incorporación de fuentes para campos de texto dinámico o de entrada" en *Utilización de Flash*.

Control de la nitidez, el grosor y el suavizado

De manera predeterminada, Flash Player o AIR determinan la configuración para los controles de la visualización del texto, como la nitidez, el grosor y el suavizado, cuando el texto cambia de tamaño, de color o se muestra con distintos fondos. En algunos casos, como cuando se tiene un texto muy pequeño o muy grande, o un texto en diversos fondos únicos, es posible que se desee controlar esta configuración. Se puede reemplazar la configuración de Flash Player o AIR mediante la clase `flash.text.TextRenderer` y sus clases asociadas, como CSMSettings. Estas clases ofrecen un control preciso de la calidad de la representación del texto incorporado. Para más información sobre las fuentes incorporadas, consulte "Utilización de fuentes incorporadas" en la página 457.

Nota: *la propiedad* `flash.text.TextField.antiAliasType` *debe tener el valor* `AntiAliasType.ADVANCED` *para que se pueda establecer la nitidez, el grosor o la propiedad gridFitType, o para utilizar el método* `TextRenderer.setAdvancedAntiAliasingTable()`.

En el ejemplo siguiente se aplican propiedades de modulación de trazo continua y formato personalizados a texto visualizado con una fuente incorporada denominada `myFont`. Cuando el usuario hace clic en el texto mostrado, Flash Player o Adobe AIR aplican la configuración personalizada:

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
    var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
    var myAliasTable:Array = new Array(myAntiAliasSettings);
    TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC,
TextColorType.DARK_COLOR, myAliasTable);
}
```

Trabajo con texto estático

El texto estático sólo se crea en la herramienta de edición. No se puede crear texto estático mediante programación con ActionScript. El texto estático resulta útil si el texto es breve y no va a cambiar (a diferencia del texto dinámico). Se puede considerar que el texto estático es un tipo de elemento gráfico, como un círculo o un cuadrado, dibujado en el escenario en la herramienta de edición de Flash. Aunque el texto estático tiene más limitaciones que el texto dinámico, ActionScript 3.0 permite leer los valores de propiedades de texto estático mediante la clase flash.text.StaticText. También se puede utilizar la clase TextSnapshot para leer valores del texto estático.

Acceso a campos de texto estático mediante la clase StaticText

Generalmente, se usa la clase flash.text.StaticText en el panel Acciones de la herramienta de edición Flash para interactuar con una instancia de texto estático colocada en el escenario. También se puede trabajar en archivos de ActionScript que interactúen con un archivo SWF que contenga texto estático. En cualquier caso, no se puede crear una instancia de texto estático mediante programación. El texto estático se crea en la herramienta de edición

Para crear una referencia a un campo de texto estático, se puede recorrer los elementos de la lista de visualización y asignar una variable. Por ejemplo:

```
for (var i = 0; i < this.numChildren; i++) {
var displayitem:DisplayObject = this.getChildAt(i);
if (displayitem instanceof StaticText) {
trace("a static text field is item " + i + " on the display list");
        var myFieldLabel:StaticText = StaticText(displayitem);
        trace("and contains the text: " + myFieldLabel.text);
}
}
```

Una vez que se tiene una referencia a un campo de texto estático, se pueden usar las propiedades de ese campo en ActionScript 3.0. El código siguiente se adjunta a un fotograma de la línea de tiempo y presupone que una variable denominada `myFieldLabel` está asignada a una referencia de texto estático. Se coloca un campo de texto dinámico denominado `myField` con respecto a los valores `x` e `y` de `myFieldLabel` y se vuelve a mostrar el valor de `myFieldLabel`.

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

Utilización de la clase TextSnapshot

Si se desea trabajar mediante programación con una instancia de texto estático existente, se puede utilizar la clase flash.text.TextSnapshot para utilizar la propiedad `textSnapshot` de un objeto flash.display.DisplayObjectContainer. Es decir, se crea una instancia de TextSnapshot a partir de la propiedad `DisplayObjectContainer.textSnapshot`. Después se pueden aplicar métodos a esa instancia para recuperar valores o seleccionar partes del texto estático.

Por ejemplo, coloque en el escenario un campo de texto estático que contenga el texto "TextSnapshot Example". Añada el siguiente código ActionScript al fotograma 1 de la línea de tiempo:

```
var mySnap:TextSnapshot = this.textSnapshot;
var count:Number = mySnap.charCount;
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

La clase TextSnapshot resulta útil para obtener el texto de campos de texto estático de un archivo SWF cargado, si se desea utilizar el texto como un valor en otra parte de una aplicación.

Ejemplo de TextField: Formato de texto con estilo periodístico

El ejemplo News Layout aplica formato a texto para que parezca una noticia de un periódico impreso. El texto de entrada puede contener un titular, un subtítulo y el cuerpo de la noticia. Dadas la anchura y la altura de visualización, el ejemplo News Layout aplica formato al titular y al subtítulo para que ocupen toda la anchura del área de visualización. El texto de la noticia se distribuye en dos o más columnas.

En este ejemplo se ilustran las siguientes técnicas de programación en ActionScript:

- Ampliar la clase TextField
- Cargar y aplicar un archivo CSS externo
- Convertir estilos de CSS en objetos TextFormat
- Utilizar la clase TextLineMetrics para obtener información sobre el tamaño de la visualización de texto

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación News Layout se encuentran en la carpeta Samples/NewsLayout. La aplicación consta de los siguientes archivos:

Archivo	Descripción
NewsLayout.mxml o NewsLayout.fla	La interfaz de usuario de la aplicación para Flex (MXML) o Flash (FLA).
com/example/programmingas3/newslayout/StoryLayoutComponent.as	Una clase UIComponent de Flex que coloca la instancia de StoryLayout.
com/example/programmingas3/newslayout/StoryLayout.as	La clase principal de ActionScript que dispone todos los componentes de una noticia para mostrarla.
com/example/programmingas3/newslayout/FormattedTextField.as	Una subclase de la clase TextField que administra su propio objeto TextFormat.
com/example/programmingas3/newslayout/HeadlineTextField.as	Una subclase de la clase FormattedTextField que ajusta el tamaño de las fuentes a una anchura deseada.
com/example/programmingas3/newslayout/MultiColumnTextField.as	Una clase de ActionScript que divide el texto en dos o más columnas.
story.css	Un archivo CSS que define estilos de texto para el diseño.

Lectura del archivo CSS externo

La aplicación News Layout empieza por leer en un archivo XML local el texto de la noticia. A continuación, lee el archivo CSS externo que proporciona la información de formato para el titular, el subtítulo y el texto principal.

El archivo CSS define tres estilos, un estilo de párrafo estándar para la noticia y los estilos h1 y h2 para el titular y el subtítulo respectivamente.

```
p {
    font-family: Georgia, "Times New Roman", Times, _serif;
    font-size: 12;
    leading: 2;
    text-align: justify;
    indent: 24;
}

h1 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
    color: #000099;
    text-align: left;
}

h2 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 16;
    font-weight: normal;
    text-align: left;
}
```

La técnica usada para leer el archivo CSS externo es la misma que la explicada en "Carga de un archivo CSS externo" en la página 455. Una vez cargado el archivo CSS, la aplicación ejecuta el método `onCSSFileLoaded()` mostrado a continuación.

```
public function onCSSFileLoaded(event:Event):void
{
    this.sheet = new StyleSheet();
    this.sheet.parseCSS(loader.data);

    h1Format = getTextStyle("h1", this.sheet);
    if (h1Format == null)
    {
        h1Format = getDefaultHeadFormat();
    }
    h2Format = getTextStyle("h2", this.sheet);
    if (h2Format == null)
    {
        h2Format = getDefaultHeadFormat();
        h2Format.size = 16;
    }
    pFormat = getTextStyle("p", this.sheet);
    if (pFormat == null)
    {
        pFormat = getDefaultTextFormat();
        pFormat.size = 12;
    }
    displayText();
}
```

El método `onCSSFileLoaded()` crea un objeto StyleSheet y hace que analice los datos CSS de entrada. El texto principal de la historia se muestra en un objeto MultiColumnTextField que puede utilizar un objeto StyleSheet directamente. No obstante, los campos de titular usan la clase HeadlineTextField, que utiliza un objeto TextFormat para su formato.

El método `onCSSFileLoaded()` llama al método `getTextStyle()` para convertir una declaración de estilos CSS en un objeto TextFormat para utilizarlo con cada uno de los dos objetos HeadlineTextField.

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
    var format:TextFormat = null;

    var style:Object = ss.getStyle(styleName);
    if (style != null)
    {
        var colorStr:String = style.color;
        if (colorStr != null && colorStr.indexOf("#") == 0)
        {
            style.color = colorStr.substr(1);
        }
        format = new TextFormat(style.fontFamily,
                        style.fontSize,
                        style.color,
                        (style.fontWeight == "bold"),
                        (style.fontStyle == "italic"),
                        (style.textDecoration == "underline"),
                        style.url,
                        style.target,
                        style.textAlign,
                        style.marginLeft,
                        style.marginRight,
                        style.indent,
                        style.leading);

        if (style.hasOwnProperty("letterSpacing"))
        {
            format.letterSpacing = style.letterSpacing;
        }
    }
    return format;
}
```

Los nombres de propiedad y el significado de los valores de propiedad difieren entre las declaraciones de estilos CSS y los objetos TextFormat. El método `getTextStyle()` traduce los valores de propiedades CSS en los valores esperados por el objeto TextFormat.

Disposición de los elementos de la noticia en la página

La clase StoryLayout aplica formato a los campos de titular, subtítulo y texto principal, y los dispone como en un periódico. El método `displayText()` crea los diversos campos y los coloca.

```
public function displayText():void
{
    headlineTxt = new HeadlineTextField(h1Format);
    headlineTxt.wordWrap = true;
    headlineTxt.x = this.paddingLeft;
    headlineTxt.y = this.paddingTop;
    headlineTxt.width = this.preferredWidth;
    this.addChild(headlineTxt);

    headlineTxt.fitText(this.headline, 1, true);

    subtitleTxt = new HeadlineTextField(h2Format);
    subtitleTxt.wordWrap = true;
    subtitleTxt.x = this.paddingLeft;
    subtitleTxt.y = headlineTxt.y + headlineTxt.height;
    subtitleTxt.width = this.preferredWidth;
    this.addChild(subtitleTxt);

    subtitleTxt.fitText(this.subtitle, 2, false);

    storyTxt = new MultiColumnText(this.numColumns, 20,
                        this.preferredWidth, 400, true, this.pFormat);
    storyTxt.x = this.paddingLeft;
    storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
    this.addChild(storyTxt);

    storyTxt.text = this.content;
...
```

Cada uno de los campos se coloca debajo del anterior estableciendo su propiedad `y` en un valor igual al de la propiedad `y` del campo anterior más su altura. Este cálculo dinámico de la posición es necesario, ya que los objetos HeadlineTextField y MultiColumnTextField pueden modificar su altura para ajustarla a su contenido.

Modificación del tamaño de fuente para ajustar el tamaño de un campo

Dados una anchura en píxeles y un número máximo de líneas para mostrar, HeadlineTextField modifica el tamaño de fuente para ajustar el texto al campo. Si el texto es corto, el tamaño de fuente es muy grande, y se creará un titular de tipo tabloide. Si el texto es largo, el tamaño de fuente es más pequeño.

El método `HeadlineTextField.fitText()` mostrado a continuación ajusta el tamaño de fuente:

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean = false,
targetWidth:Number = -1):uint
{
    this.text = toUpper ? msg.toUpperCase() : msg;

    if (targetWidth == -1)
    {
        targetWidth = this.width;
    }

    var pixelsPerChar:Number = targetWidth / msg.length;

    var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar * 1.8 * maxLines));

    if (pointSize < 6)
    {
        // the point size is too small
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(--pointSize, maxLines);
    }
    else
    {
        return growText(pointSize, maxLines);
    }
}

public function growText(pointSize:Number, maxLines:uint = 1):Number
{
    if (pointSize >= MAX_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
        // set it back to the last size
        this.changeSize(pointSize);
        return pointSize;
    }
    else
    {
        return growText(pointSize + 1, maxLines);
    }
```

```
}

public function shrinkText(pointSize:Number, maxLines:uint=1):Number
{
    if (pointSize <= MIN_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
        return pointSize;
    }
}
```

El método `HeadlineTextField.fitText()` utiliza una técnica recursiva simple para ajustar el tamaño de la fuente. Primero estima un número de píxeles por carácter en el texto y con ese dato calcula un tamaño de punto inicial. A continuación cambia el tamaño de la fuente y comprueba si el texto se ha ajustado para crear un número de líneas de texto superior al máximo. Si hay demasiadas líneas, llama al método `shrinkText()` para reducir el tamaño de fuente y lo intenta de nuevo. Si no hay demasiadas líneas, llama al método `growText()` para aumentar el tamaño de fuente y lo intenta de nuevo. El proceso se detiene en el punto en el que, al aumentar un punto el tamaño de fuente, se crearían demasiadas líneas.

División del texto en varias columnas

La clase MultiColumnTextField divide el texto en varios objetos TextField que se disponen como las columnas de un periódico.

El constructor `MultiColumnTextField()` crea primero un conjunto de objetos TextField, uno por cada columna, como se muestra a continuación:

```
    for (var i:int = 0; i < cols; i++)
    {
        var field:TextField = new TextField();
        field.multiline = true;
        field.autoSize = TextFieldAutoSize.NONE;
        field.wordWrap = true;
        field.width = this.colWidth;
        field.setTextFormat(this.format);
        this.fieldArray.push(field);
        this.addChild(field);
    }
```

Se añade cada objeto TextField al conjunto y a la lista de visualización con el método `addChild()`.

Siempre que cambien las propiedades `text` y `styleSheet` del objeto StoryLayout, se llama al método `layoutColumns()` para volver a mostrar el texto. El método `layoutColumns()` llama al método `getOptimalHeight()` para estimar la altura en píxeles necesaria para ajustar todo el texto en la anchura de diseño especificada.

```
public function getOptimalHeight(str:String):int
{
    if (field.text == "" || field.text == null)
    {
        return this.preferredHeight;
    }
    else
    {
        this.linesPerCol = Math.ceil(field.numLines / this.numColumns);

        var metrics:TextLineMetrics = field.getLineMetrics(0);
        this.lineHeight = metrics.height;
        var prefHeight:int = linesPerCol * this.lineHeight;

        return prefHeight + 4;
    }
}
```

En primer lugar, el método `getOptimalHeight()` calcula la anchura de cada columna. A continuación establece la anchura y la propiedad `htmlText` del primer objeto TextField del conjunto. El método `getOptimalHeight()` utiliza ese primer objeto TextField para descubrir el número total de líneas ajustadas en el texto y, a partir de ese número, determina cuántas líneas debe haber en cada columna. A continuación, llama al método `TextField.getLineMetrics()` para recuperar un objeto TextLineMetrics que contiene detalles sobre el tamaño del texto en la primera línea. La propiedad `TextLineMetrics.height` representa la altura de una línea de texto, en píxeles, incluidos los valores ascendente, descendente y de interlineado. La altura óptima para el objeto MultiColumnTextField es la altura de línea multiplicada por el número de líneas por columna, más 4 para tener en cuenta el borde de dos píxeles que hay por encima y por debajo de un objeto TextField.

A continuación se muestra el código del método `layoutColumns()`:

```
public function layoutColumns():void
{
    if (this._text == "" || this._text == null)
    {
        return;
    }

    var field:TextField = fieldArray[0] as TextField;
    field.text = this._text;
    field.setTextFormat(this.format);

    this.preferredHeight = this.getOptimalHeight(field);

    var remainder:String = this._text;
    var fieldText:String = "";
    var lastLineEndedPara:Boolean = true;

    var indent:Number = this.format.indent as Number;

    for (var i:int = 0; i < fieldArray.length; i++)
    {
        field = this.fieldArray[i] as TextField;

        field.height = this.preferredHeight;
        field.text = remainder;

        field.setTextFormat(this.format);
```

```
var lineLen:int;
if (indent > 0 && !lastLineEndedPara && field.numLines > 0)
{
    lineLen = field.getLineLength(0);
    if (lineLen > 0)
    {
        field.setTextFormat(this.firstLineFormat, 0, lineLen);
        }
    }

field.x = i * (colWidth + gutter);
field.y = 0;

remainder = "";
fieldText = "";

var linesRemaining:int = field.numLines;
var linesVisible:int = Math.min(this.linesPerCol, linesRemaining);

for (var j:int = 0; j < linesRemaining; j++)
{
    if (j < linesVisible)
    {
        fieldText += field.getLineText(j);
    }
    else
    {
        remainder +=field.getLineText(j);
    }
}

field.text = fieldText;

field.setTextFormat(this.format);

if (indent > 0 && !lastLineEndedPara)
{
    lineLen = field.getLineLength(0);
    if (lineLen > 0)
    {
        field.setTextFormat(this.firstLineFormat, 0, lineLen);
    }
}

var lastLine:String = field.getLineText(field.numLines - 1);
var lastCharCode:Number = lastLine.charCodeAt(lastLine.length - 1);
```

```
        if (lastCharCode == 10 || lastCharCode == 13)
        {
        lastLineEndedPara = true;
        }
        else
        {
        lastLineEndedPara = false;
        }

        if ((this.format.align == TextFormatAlign.JUSTIFY) &&
                (i < fieldArray.length - 1))
        {
        if (!lastLineEndedPara)
        {
            justifyLastLine(field, lastLine);
        }
    }
    }
}
```

Una vez establecida la propiedad `preferredHeight` llamando al método `getOptimalHeight()`, el método `layoutColumns()` recorre los objetos TextField y establece la altura de cada uno de ellos en el valor de `preferredHeight`. A continuación, el método `layoutColumns()` distribuye a cada campo las líneas de texto suficientes para que no se produzca desplazamiento en ningún campo y el texto de cada campo empiece donde acabó el texto del campo anterior. Si el estilo de alineación del texto está establecido en "justify" (justificado), se llama al método `justifyLastLine()` para justificar la última línea del texto de un campo. De lo contrario, esa última línea se trataría como una línea de final de párrafo y no se justificaría.

Utilización de Flash Text Engine

Flash Text Engine (FTE) ofrece una compatibilidad de bajo nivel para realizar un sofisticado control de las medidas y el formato del texto, además de admitir texto bidireccional. Ofrece una compatibilidad de idiomas y un flujo de texto mejorados. Aunque se puede usar para crear y administrar elementos de texto sencillos, el FTE está diseñado principalmente como base sobre la que los desarrolladores puedan crear componentes de gestión del texto. Como tal, Flash Text Engine presupone un nivel más avanzado de conocimientos de programación. Para mostrar elementos de texto sencillos, consulte las secciones anteriores que explican el uso del objeto `TextField` y de otros objetos relacionados.

Nota: *Flash Text Engine está disponible a partir de Flash Player 10 y Adobe AIR 1.5.*

Creación y visualización de texto

Las clases que conforman Flash Text Engine permiten crear texto, aplicarle formato y controlarlo. Las clases siguientes son los componentes esenciales para crear y mostrar texto con Flash Text Engine:

- TextElement/GraphicElement/GroupElement; incluyen el contenido de una instancia de TextBlock
- ElementFormat; especifica los atributos de formato para el contenido de una instancia de TextBlock
- TextBlock; componente fundamental para crear un párrafo de texto
- TextLine; línea de texto creada a partir del objeto TextBlock

Para mostrar el texto, cree un objeto TextElement a partir de una cadena y de un objeto ElementFormat, que especifica las características de formato, y asigne el objeto TextElement a la propiedad `content` de un objeto TextBlock. Las líneas de texto que se van a mostrar se crean llamando al método `TextBlock.createTextLine()`. Por ejemplo, el código siguiente utiliza estas clases de FTE para mostrar el texto "Hello World! This is Flash Text Engine!" usando el formato y los valores de fuente predeterminados.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class HelloWorldExample extends Sprite
    {
        public function HelloWorldExample()
        {
            var str = "Hello World! This is Flash Text Engine!";
            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;

            var textLine1:TextLine = textBlock.createTextLine(null, 300);
            addChild(textLine1);
            textLine1.x = 30;
            textLine1.y = 30;
        }
    }
}
```

Los parámetros para `createTextLine()` especifican la línea a partir de la que comenzar la nueva línea y la anchura de la línea en píxeles. La línea a partir de la que comenzar la nueva línea suele ser la línea anterior pero, en caso de que sea la primera línea, es `null`.

Adición de objetos GraphicElement y GroupElement

Para mostrar una imagen o un elemento gráfico, se puede asignar un objeto GraphicElement a un objeto TextBlock. Sólo hay que crear una instancia de la clase GraphicElement a partir de un gráfico o una imagen y asignar la instancia a la propiedad `TextBlock.content`. La línea de texto se crea de la manera habitual, llamando a `TextBlock.createTextline()`. El ejemplo siguiente crea dos líneas de texto, una con un objeto GraphicElement y otra con un objeto TextElement.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GraphicElementExample extends Sprite
    {
        public function GraphicElementExample()
        {
            var str:String = "Beware of Dog!";

            var triangle:Shape = new Shape();
            triangle.graphics.beginFill(0xFF0000, 1);
            triangle.graphics.lineStyle(3);
            triangle.graphics.moveTo(30, 0);
            triangle.graphics.lineTo(60, 50);
            triangle.graphics.lineTo(0, 50);
            triangle.graphics.lineTo(30, 0);
            triangle.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;

            var graphicElement:GraphicElement = new GraphicElement(triangle, triangle.width,
triangle.height, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = graphicElement;
            var textLine1:TextLine = textBlock.createTextLine(null, triangle.width);
            textLine1.x = 50;
            textLine1.y = 110;
            addChild(textLine1);

            var textElement:TextElement = new TextElement(str, format);
            textBlock.content = textElement;
            var textLine2 = textBlock.createTextLine(null, 300);
            addChild(textLine2);
            textLine2.x = textLine1.x - 30;
            textLine2.y = textLine1.y + 15;
        }
    }
}
```

Se puede crear un objeto GroupElement para crear un grupo de objetos TextElement, GraphicElement y otros objetos de tipo GroupElement que asignar a la propiedad `content` de un objeto TextBlock. El parámetro pasado al constructor `GroupElement()` es un Vector, que apunta a los elementos de texto, gráficos y grupo que conforman el grupo. El ejemplo siguiente agrupa dos elementos gráficos y un elemento de texto, y los asigna como una unidad a un bloque de texto.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GroupElementExample extends Sprite
    {
        public function GroupElementExample()
        {
            var str:String = "Beware of Alligators!";

            var triangle1:Shape = new Shape();
            triangle1.graphics.beginFill(0xFF0000, 1);
            triangle1.graphics.lineStyle(3);
            triangle1.graphics.moveTo(30, 0);
            triangle1.graphics.lineTo(60, 50);
            triangle1.graphics.lineTo(0, 50);
            triangle1.graphics.lineTo(30, 0);
            triangle1.graphics.endFill();

            var triangle2:Shape = new Shape();
            triangle2.graphics.beginFill(0xFF0000, 1);
            triangle2.graphics.lineStyle(3);
            triangle2.graphics.moveTo(30, 0);
            triangle2.graphics.lineTo(60, 50);
            triangle2.graphics.lineTo(0, 50);
            triangle2.graphics.lineTo(30, 0);
            triangle2.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;
            var graphicElement1:GraphicElement = new GraphicElement(triangle1,
triangle1.width, triangle1.height, format);
            var textElement:TextElement = new TextElement(str, format);
            var graphicElement2:GraphicElement = new GraphicElement(triangle2,
triangle2.width, triangle2.height, format);
            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>();
            groupVector.push(graphicElement1, textElement, graphicElement2);
            var groupElement = new GroupElement(groupVector);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = groupElement;
            var textLine:TextLine = textBlock.createTextLine(null, 800);
            addChild(textLine);
            textLine.x = 100;
            textLine.y = 200;
        }
    }
}
```

Reemplazo de texto

El texto de una instancia de TextBlock se puede reemplazar llamando al método `TextElement.replaceText()` para reemplazar el texto del elemento TextElement que asignó a la propiedad `TextBlock.content`. El ejemplo siguiente utiliza `replaceText()` para insertar texto al principio de la línea, añadir texto al final de la línea y reemplazar texto en medio de la línea.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class ReplaceTextExample extends Sprite
    {
        public function ReplaceTextExample()
        {

            var str:String = "Lorem ipsum dolor sit amet";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription);
            format.fontSize = 14;
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;
            createLine(textBlock, 10);
            textElement.replaceText(0, 0, "A text fragment: ");
            createLine(textBlock, 30);
            textElement.replaceText(43, 43, "...");
            createLine(textBlock, 50);
            textElement.replaceText(23, 28, "(ipsum)");
            createLine(textBlock, 70);
        }

        function createLine(textBlock:TextBlock, y:Number):void {
            var textLine:TextLine = textBlock.createTextLine(null, 300);
            textLine.x = 10;
            textLine.y = y;
            addChild(textLine);
        }
    }
}
```

El método `replaceText()` reemplaza el texto especificado por los parámetros `beginIndex` y `endIndex` con el texto especificado por el parámetro `newText`. Si los valores de los parámetros `beginIndex` y `endIndex` son iguales, `replaceText()` inserta el texto especificado en esa ubicación. De lo contrario, reemplaza los caracteres especificados por `beginIndex` y `endIndex` con el nuevo texto.

Gestión de eventos en FTE

Se pueden añadir detectores de eventos a una instancia de TextLine igual que ocurre con otros objetos de visualización. Por ejemplo, se puede detectar cuándo un usuario pasa el ratón por encima de una línea de texto o cuándo hace clic en la línea. El ejemplo siguiente detecta los dos eventos mencionados. Cuando se pasa el ratón por encima de la línea, el cursor cambia a un cursor de botón y, cuando se hace clic en la línea, ésta cambia de color.

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventHandlerExample extends Sprite
    {
        var textBlock:TextBlock = new TextBlock();

        public function EventHandlerExample():void
        {
            var str:String = "I'll change color if you click me.";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription, 18);
            var textElement = new TextElement(str, format);
            textBlock.content = textElement;
            createLine(textBlock);
        }

        private function createLine(textBlock:TextBlock):void
        {
            var textLine:TextLine = textBlock.createTextLine(null, 500);
            textLine.x = 30;
            textLine.y = 30;
            addChild(textLine);
            textLine.addEventListener("mouseOut", mouseOutHandler);
            textLine.addEventListener("mouseOver", mouseOverHandler);
            textLine.addEventListener("click", clickHandler);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
            Mouse.cursor = "arrow";
        }

        function clickHandler(event:MouseEvent):void {
            if(textBlock.firstLine)
                removeChild(textBlock.firstLine);
            var newFormat:ElementFormat = textBlock.content.elementFormat.clone();
```

```
            switch(newFormat.color)
            {
                case 0x000000:
                    newFormat.color = 0xFF0000;
                    break;
                case 0xFF0000:
                    newFormat.color = 0x00FF00;
                    break;
                case 0x00FF00:
                    newFormat.color = 0x0000FF;
                    break;
                case 0x0000FF:
                    newFormat.color = 0x000000;
                    break;
            }
            textBlock.content.elementFormat = newFormat;
            createLine(textBlock);
        }
    }
}
```

Reflejo de eventos

También se pueden reflejar los eventos de un bloque de texto, o de una parte de un bloque de texto, en un distribuidor de eventos. En primer lugar, se crea una instancia de EventDispatcher y después se asigna a la propiedad `eventMirror` de una instancia de TextElement. Si el bloque de texto consta de un solo elemento de texto, el motor de texto refleja los eventos de todo el bloque de texto. Si el bloque de texto está formado por varios elementos de texto, el motor de texto sólo refleja los eventos de las instancias de TextElement que tienen establecida la propiedad `eventMirror`. El texto del ejemplo siguiente consta de tres elementos: la palabra "Click", la palabra "here" y la cadena "to see me in italic". El ejemplo asigna un distribuidor de eventos al segundo elemento de texto, la palabra "here", y añade un detector de eventos, el método `clickHandler()`. El método `clickHandler()` aplica al texto formato de cursiva. Además, reemplaza el contenido del tercer elemento de texto para que se lea, "Click me here to see me in normal font!".

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventMirrorExample extends Sprite
    {
        var fontDescription:FontDescription = new FontDescription("Helvetica", "bold");
        var format:ElementFormat = new ElementFormat(fontDescription, 18);
        var textElement1 = new TextElement("Click ", format);
        var textElement2 = new TextElement("here ", format);
        var textElement3 = new TextElement("to see me in italic! ", format);
        var textBlock:TextBlock = new TextBlock();

        public function EventMirrorExample()
        {
            var myEvent:EventDispatcher = new EventDispatcher();

            myEvent.addEventListener("click", clickHandler);
            myEvent.addEventListener("mouseOut", mouseOutHandler);
```

```
            myEvent.addEventListener("mouseOver", mouseOverHandler);

            textElement2.eventMirror=myEvent;

            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>;
            groupVector.push(textElement1, textElement2, textElement3);
            var groupElement:GroupElement = new GroupElement(groupVector);

            textBlock.content = groupElement;
            createLines(textBlock);
        }

        private function clickHandler(event:MouseEvent):void
        {
            var newFont:FontDescription = new FontDescription();
            newFont.fontWeight = "bold";

            var newFormat:ElementFormat = new ElementFormat();
            newFormat.fontSize = 18;
            if(textElement3.text == "to see me in italic! ") {
                newFont.fontPosture = FontPosture.ITALIC;
                textElement3.replaceText(0,21, "to see me in normal font! ");
            }
            else {
                newFont.fontPosture = FontPosture.NORMAL;
                textElement3.replaceText(0, 26, "to see me in italic! ");
            }
            newFormat.fontDescription = newFont;
            textElement1.elementFormat = newFormat;
            textElement2.elementFormat = newFormat;
            textElement3.elementFormat = newFormat;
            createLines(textBlock);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
                Mouse.cursor = "arrow";
        }

        private function createLines(textBlock:TextBlock):void
        {
            if(textBlock.firstLine)
                removeChild (textBlock.firstLine);
            var textLine:TextLine = textBlock.createTextLine (null, 300);
            textLine.x = 15;
            textLine.y = 20;
            addChild (textLine);
        }
    }
}
```

Las funciones `mouseOverHandler()` y `mouseOutHandler()` cambian el cursor a un cursor de botón cuando se encuentra sobre la palabra "here" y le devuelven la forma de flecha cuando no lo está.

Formato de texto

Un objeto `TextBlock` es la base para crear líneas de texto. El contenido de un `TextBlock` se asigna mediante el objeto `TextElement`. Un objeto `ElementFormat` controla el formato del texto. La clase ElementFormat define propiedades como la alineación de la línea de base, el kerning (ajuste entre caracteres), el seguimiento, la rotación de texto y el tamaño, el color y el formato de mayúsculas de la fuente. También incluye un elemento `FontDescription`, que se analiza detalladamente en "Trabajo con fuentes" en la página 479.

Utilización del objeto ElementFormat

El constructor del objeto `ElementFormat` toma cualquiera de una larga lista de parámetros opcionales, incluido el parámetro `FontDescription`. Estas propiedades también se pueden establecer desde fuera del constructor. El ejemplo siguiente indica la relación de los distintos objetos en la definición y visualización de una línea de texto sencillo:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function ElementFormatExample()
        {
            fd.fontName = "Garamond";
            ef = new ElementFormat(fd);
            ef.fontSize = 30;
            ef.color = 0xFF0000;
            str = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

Color de fuente y transparencia (alfa)

La propiedad `color` del objeto `ElementFormat` define el color de la fuente. El valor es un entero que representa los componentes RGB del color; por ejemplo, 0xFF0000 para el rojo y 0x00FF00 para el verde. El valor predeterminado es el negro (0x000000).

La propiedad `alpha` define el valor de transparencia alfa de un elemento (de `TextElement` y de `GraphicElement`). Los valores se encuentran entre 0 (totalmente transparente) y 1 (completamente opaco, que es el valor predeterminado). Los elementos que tienen un valor `alpha` de 0 son invisibles, pero siguen estando activos. Este valor se multiplica por los valores de alpha heredados, haciendo así que el elemento sea más transparente.

```
var ef:ElementFormat = new ElementFormat();
ef.alpha = 0.8;
ef.color = 0x999999;
```

Alineación y desplazamiento de la línea de base

La fuente y el tamaño del texto más grande de una línea determinan su línea de base dominante. Estos valores se pueden reemplazar definiendo las propiedades `TextBlock.baselineFontDescription` y `TextBlock.baselineFontSize`. La línea de base dominante se puede alinear con una de las diversas líneas de base del texto. Entre ellas están la línea ascendente y la línea descendente, o la línea superior, central o inferior de los pictogramas.

***A.** Ascendente **B.** Línea de base **C.** Descendente **D.** x-height (altura x)*

En el objeto `ElementFormat`, las características de línea de base y alineación se definen mediante tres propiedades. La propiedad `alignmentBaseline` establece la línea de base principal de un objeto `TextElement` o `GraphicElement`. Esta línea de base es la línea "de ajuste" para el elemento y respecto a su posición se alinea la línea de base dominante de todo el texto.

La propiedad `dominantBaseline` especifica cuál de las distintas líneas de base del elemento se debe usar, lo que determina la posición vertical del elemento en la línea. El valor predeterminado es `TextBaseline.ROMAN`, pero también se pueden establecer como dominantes las líneas de base `IDEOGRAPHIC_TOP` o `IDEOGRAPHIC_BOTTOM`.

La propiedad `baselineShift` desplaza la línea de base un número determinado de píxeles a lo largo del eje y. En el caso de texto normal (no rotado), un valor positivo desplaza la línea de base hacia abajo, mientras que un valor negativo lo hace hacia arriba..

Formato tipográfico de mayúsculas y minúsculas

La propiedad `TypographicCase` de `ElementFormat` especifica el uso de mayúsculas, minúsculas o versalitas del texto.

```
var ef_Upper:ElementFormat = new ElementFormat();
ef_Upper.typographicCase = TypographicCase.UPPERCASE;

var ef_SmallCaps:ElementFormat = new ElementFormat();
ef_SmallCaps.typographicCase = TypographicCase.SMALL_CAPS;
```

Rotación del texto

Los bloques de texto o los pictogramas incluidos en un segmento de texto se pueden rotar en incrementos de 90 grados. La clase TextRotation define las constantes siguientes para establecer la rotación de los bloques de texto y los pictogramas:

Constante	Valor	Descripción
AUTO	"auto"	Especifica una rotación de 90 grados en sentido contrario a las agujas del reloj. Se utiliza normalmente con textos asiáticos, para rotar sólo los pictogramas que necesitan rotarse.
ROTATE_0	"rotate_0"	No especifica ninguna rotación.
ROTATE_180	"rotate_180"	Especifica una rotación de 180 grados.
ROTATE_270	"rotate_270"	Especifica una rotación de 270 grados.
ROTATE_90	"rotate_90"	Especifica una rotación de 90 grados en el sentido de las agujas del reloj.

Para rotar las líneas de texto de un bloque de texto, establezca la propiedad `TextBlock.lineRotation` antes de llamar al método `TextBlock.createTextLine()` para crear la línea de texto.

Para rotar los pictogramas incluidos en un bloque de texto o un segmento, establezca la propiedad `ElementFormat.textRotation` en el número de grados que desea girar los pictogramas. Un pictograma es la forma que constituye un carácter, o una parte de un carácter que consta de varios pictogramas. La letra a y el punto de la i, por ejemplo, son pictogramas.

Rotar pictogramas es relevante en algunos idiomas asiáticos en los que puede ser necesario rotar las líneas hasta el sentido vertical, pero no los caracteres incluidos en las líneas. Para más información sobre la rotación de texto en idiomas asiáticos, consulte "Justificación de texto asiático" en la página 483.

El siguiente es un ejemplo de rotación tanto de un bloque de texto como de los pictogramas que incluye, tal y como se haría con textos asiáticos. El ejemplo también utiliza una fuente japonesa:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class RotationExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function RotationExample()
        {
            fd.fontName = "MS Mincho";
            ef = new ElementFormat(fd);
            ef.textRotation = TextRotation.AUTO;
            str = "This is rotated Japanese text";
            te = new TextElement(str, ef);
            tb.lineRotation = TextRotation.ROTATE_90;
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

Bloqueo y clonación de ElementFormat

Cuando se asigna un objeto `ElementFormat` a un tipo de `ContentElement`, su propiedad `locked` se establece automáticamente en `true`. Si se intenta modificar un objeto `ElementFormat` bloqueado, se genera un error de tipo `IllegalOperationError`. La práctica más recomendable consiste en definir completamente un objeto así antes de asignarlo a una instancia de `TextElement`.

Si se desea modificar una instancia de `ElementFormat`, se debe comprobar antes su propiedad `locked`. Si es `true`, utilice el método `clone()` para crear una copia no bloqueada del objeto. Las propiedades de este objeto desbloqueado se pueden cambiar y, posteriormente, asignarlo a la instancia de `TextElement`. Las nuevas líneas que se creen a partir de él tendrán el nuevo formato. Las líneas anteriores creadas a partir de este mismo objeto y con el formato antiguo no se modifican.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd:FontDescription = new FontDescription();

        public function ElementFormatCloneExample()
        {
            fd.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            ef1.fontSize = 24;
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontSize = 32;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

Trabajo con fuentes

El objeto `FontDescription` se utiliza conjuntamente con `ElementFormat` para identificar una fuente y definir algunas de sus características. Entre estas características se encuentran el nombre de la fuente, su grosor, su postura, su representación y el modo de localizar la fuente (en el dispositivo en lugar de incorporada).

Nota: *FTE no admite fuentes de Tipo 1 o fuentes de mapa de bits, como las de Tipo 3, ATC, sfnt-wrapped CID o Naked CID.*

Definición de características de fuentes (objeto FontDescription)

La propiedad `fontName` del objeto `FontDescription` puede ser un único nombre o una lista de nombres separadas por comas. Por ejemplo, en una lista como "Arial, Helvetica, _sans", Flash Player o AIR buscan primero "Arial", después "Helvetica" y, por último, "_sans", si no encuentran alguna de las dos primeras fuentes. El conjunto de nombres de fuente incluye tres nombres genéricos de fuentes de dispositivo: "_sans", "_serif" y "_typewriter". Dependiendo del sistema de reproducción, se corresponden con fuentes de dispositivo específicas. Es recomendable especificar nombres predeterminados como estos en todas las descripciones de fuentes que utilizan fuentes de dispositivo. Si no se especifica `fontName`, se utiliza "_serif" de manera predeterminada.

La propiedad `fontPosture` se puede establecer en su valor predeterminado (`FontPosture.NORMAL`) o en cursiva (`FontPosture.ITALIC`). La propiedad `fontWeight` se puede establecer en su valor predeterminado (`FontWeight.NORMAL`) o en negrita (`FontWeight.BOLD`).

```
var fd1:FontDescription = new FontDescription();
fd1.fontName = "Arial, Helvetica, _sans";
fd1.fontPosture = FontPosture.NORMAL;
fd1.fontWeight = FontWeight.BOLD;
```

Fuentes incorporadas frente a fuentes de dispositivo

La propiedad `fontLookup` del objeto `FontDescription` especifica si Flash Player y AIR deben buscar una fuente de dispositivo o una fuente incorporada para representar el texto. Si se especifica una fuente de dispositivo (`FontLookup.DEVICE`), el tiempo de ejecución buscará la fuente en el sistema de reproducción. La especificación de una fuente incorporada (`FontLookup.EMBEDDED_CFF`) hace que el tiempo de ejecución busque una fuente incorporada con el nombre especificado en el archivo SWF. Con esta configuración sólo funcionan las fuentes CFF (de formato de fuente compacto) incorporadas. Si no se encuentra la fuente especificada, se utiliza una fuente de dispositivo alternativa.

Las fuentes de dispositivo suponen un tamaño de archivo SW menor. Las fuentes incorporadas ofrecen una mayor fidelidad entre palataformas.

```
var fd1:FontDescription = new FontDescription();
fd1.fontLookup = FontLookup.EMBEDDED_CFF;
fd1.fontName = "Garamond, _serif";
```

Modo de representación e interpolación

El procesado CFF (Formato de fuentes compactas - Compact Font Format) está disponible a partir de Flash Player 10 y Adobe AIR 1.5. Este tipo de procesado de fuentes hace que el texto sea más legible y permite una visualización de mayor calidad en fuentes de tamaño pequeño. Esta configuración sólo se aplica a las fuentes incorporadas. `FontDescription` toma como predeterminada esta configuración (`RenderingMode.CFF`) para la propiedad `renderingMode` . Esta propiedad se puede establecer en `RenderingMode.NORMAL` para que coincida con el tipo de representación utilizado por Flash Player 7 o versiones anteriores.

Cuando está seleccionada la representación CFF, una segunda propiedad, `cffHinting`, controla cómo se ajustan los trazos horizontales de una fuente a la cuadrícula de subpíxeles. El valor predeterminado, `CFFHinting.HORIZONTAL_STEM`, utiliza la interpolación CFF. Estableciendo esta propiedad en `CFFHinting.NONE` se elimina la interpolación, lo que es adecuado para animaciones o para tamaños de fuente grandes.

```
var fd1:FontDescription = new FontDescription();
fd1.renderingMode = RenderingMode.CFF;
fd1.cffHinting = CFFHinting.HORIZONTAL_STEM;
```

Bloqueo y clonación de FontDescription

Cuando un objeto `FontDescription` está asignado a un `ElementFormat`, su propiedad `locked` se establece automáticamente en `true`. Si se intenta modificar un objeto `FontDescription` bloqueado, se genera un error de tipo `IllegalOperationError`. La práctica más recomendable consiste en definir completamente un objeto así antes de asignarlo a una instancia de `ElementFormat`.

Si se desea modificar una `FontDescription` existente, se debe comprobar antes su propiedad `locked`. Si es `true`, utilice el método `clone()` para crear una copia no bloqueada del objeto. Las propiedades de este objeto desbloqueado se pueden cambiar y, posteriormente, asignarlo a `ElementFormat`. Las nuevas líneas que se creen a partir de este `TextElement` tendrán el nuevo formato. Las líneas anteriores creadas a partir de este mismo objeto no cambian.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class FontDescriptionCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd1:FontDescription = new FontDescription();
        private var fd2:FontDescription;

        public function FontDescriptionCloneExample()
        {
            fd1.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            fd2 = (fd1.locked) ? fd1.clone() : fd1;
            fd2.fontName = "Arial";
            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontDescription = fd2;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

Control del texto

FTE ofrece un nuevo conjunto de controles de formato del texto para manejar la justificación y el espaciado (manual y entre caracteres). También hay propiedades para controlar si hay líneas rotas y para establecer tabulaciones dentro de las líneas.

Justificación del texto

Si se justifica el texto, se da la misma longitud a todas las líneas de un párrafo ajustando el espaciado entre las palabras y, a veces, entre las letras. El efecto consiste en alinear el texto a ambos lados, mientras que se varía el espaciado entre las palabras y las letras. Las columnas de texto de los periódicos y las revistas suelen estar justificadas.

La propiedad `lineJustfication` de la clase SpaceJustifier permite controlar la justificación de las líneas de un bloque de texto. La clase LineJustification define constantes que se pueden usar para especificar una opción de justificación: `ALL_BUT_LAST` justifica todas las líneas de texto excepto la última; `ALL_INCLUDING_LAST` justifica todo el texto, incluida la última línea; `UNJUSTIFIED`, que es la constante predeterminada, deja el texto sin justificar.

Para justificar el texto se debe establecer la propiedad `lineJustification` en una instancia de la clase SpaceJustifier y asignar esa instancia a la propiedad `textJustifier` de una instancia de TextBlock. El ejemplo siguiente crea un párrafo en el que todas las líneas de texto están justificadas, excepto la última.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class JustifyExample extends Sprite
    {
        public function JustifyExample()
        {
            var str:String = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, " +
            "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut " +
            "enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut " +
            "aliquip ex ea commodo consequat.";

            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement=new TextElement(str,format);
            var spaceJustifier:SpaceJustifier=new
SpaceJustifier("en",LineJustification.ALL_BUT_LAST);

            var textBlock:TextBlock = new TextBlock();
            textBlock.content=textElement;
            textBlock.textJustifier=spaceJustifier;
            createLines(textBlock);
        }

        private function createLines(textBlock:TextBlock):void {
            var yPos=20;
            var textLine:TextLine=textBlock.createTextLine(null,150);

            while (textLine) {
                addChild(textLine);
                textLine.x=15;
                yPos+=textLine.textHeight+2;
                textLine.y=yPos;
                textLine=textBlock.createTextLine(textLine,150);
            }
        }
    }
}
```

Para variar el espaciado entre las letras además de entre las palabras, se establece la propiedad `SpaceJustifier.letterspacing` en `true`. Activando el espaciado entre las letras se puede reducir la aparición de antiestéticos espacios entre las palabras, lo que a veces ocurre con la justificación simple.

Justificación de texto asiático

Justificar textos en idiomas de Extremo Oriente implica otras consideraciones. Se puede escribir de arriba a abajo y algunos caracteres, conocidos como kinsoku, no pueden aparecer al principio o al final de una línea. La clase JustificationStyle define las siguientes constantes, que especifican las opciones para gestionar estos caracteres. `PRIORITIZE_LEAST_ADJUSTMENT` basa la justificación en expandir o comprimir la línea, dependiendo de qué opción ofrece el resultado más deseable. `PUSH_IN_KINSOKU` basa la justificación en comprimir los kinsoku al final de la línea, o en expandirla si no se produce kinsoku, o bien, si ese espacio es insuficiente.

`PUSH_OUT_ONLY`; la justificación se realiza expandiendo la línea. Para crear un bloque de texto asiático vertical, establezca la propiedad `TextBlock.lineRotation` en `TextRotation.ROTATE_90` y la propiedad `ElementFormat.textRotation`, en `TextRotation.AUTO`, que es el valor predeterminado. Si se establece la propiedad `textRotation` en `AUTO`, los pictogramas del texto se mantienen en vertical en lugar de girarse sobre un lado cuando se rota la línea. El valor `AUTO` rota 90 grados en sentido contrario al de las agujas del reloj sólo los pictogramas de anchura completa y los anchos, según lo determinen las propiedades Unicode del pictograma. El ejemplo siguiente muestra un bloque vertical de texto en japonés y lo justifica utilizando la opción `PUSH_IN_KINSOKU`. i

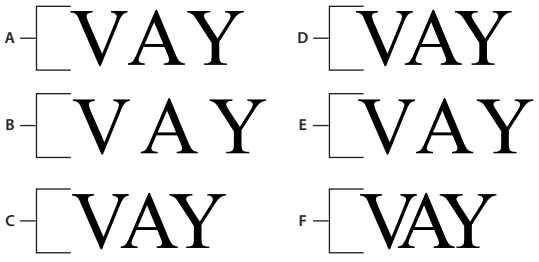
```
package
{
    import flash.text.engine.*;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.system.Capabilities;

    public class EastAsianJustifyExample extends Sprite
    {
        public function EastAsianJustifyExample()
        {
            var Japanese_txt:String = String.fromCharCode(
            0x5185, 0x95A3, 0x5E9C, 0x304C, 0x300C, 0x653F, 0x5E9C, 0x30A4,
            0x30F3, 0x30BF, 0x30FC, 0x30CD, 0x30C3, 0x30C8, 0x30C6, 0x30EC,
            0x30D3, 0x300D, 0x306E, 0x52D5, 0x753B, 0x914D, 0x4FE1, 0x5411,
            0x3051, 0x306B, 0x30A2, 0x30C9, 0x30D3, 0x30B7, 0x30B9, 0x30C6,
            0x30E0, 0x30BA, 0x793E, 0x306E)
            var textBlock:TextBlock = new TextBlock();
            var font:FontDescription = new FontDescription();
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 12;
            format.color = 0xCC0000;
            format.textRotation = TextRotation.AUTO;
            textBlock.baselineZero = TextBaseline.IDEOGRAPHIC_CENTER;
            var eastAsianJustifier:EastAsianJustifier = new EastAsianJustifier("ja",
LineJustification.ALL_BUT_LAST);
            eastAsianJustifier.justificationStyle = JustificationStyle.PUSH_IN_KINSOKU;
            textBlock.textJustifier = eastAsianJustifier;
            textBlock.lineRotation = TextRotation.ROTATE_90;
            var linePosition:Number = this.stage.stageWidth - 75;
            if (Capabilities.os.search("Mac OS") > -1)
                // set fontName: Kozuka Mincho Pro R
```

```
                font.fontName = String.fromCharCode(0x5C0F, 0x585A, 0x660E, 0x671D) + " Pro R";
            else
                font.fontName = "Kozuka Mincho Pro R";
            textBlock.content = new TextElement(Japanese_txt, format);
            var previousLine:TextLine = null;

            while (true)
            {
                var textLine:TextLine = textBlock.createTextLine(previousLine, 200);
                if (textLine == null)
                    break;
                textLine.y = 20;
                textLine.x = linePosition;
                linePosition -= 25;
                addChild(textLine);
                previousLine = textLine;
            }
        }
    }
}
```

Espaciado manual y entre caracteres (kerning)

El espaciado manual y entre caracteres (kerning) afectan a la distancia existente entre pares adyacentes de caracteres de un bloque de texto. El kerning (espaciado entre caracteres) controla cómo se "ajustan" entre sí los pares de caracteres, como por ejemplo "WA" o "Va". El espaciado manual se establece en el objeto `ElementFormat`. De forma predeterminada está activado (`Kerning.ON`) y se puede establecer en OFF o en AUTO, en cuyo caso el espaciado manual sólo se aplica entre los caracteres si ninguno de ellos es Kanji, Hiragana o Katakana.

El espaciado manual añade o retira un número determinado de píxeles entre todos los caracteres de un bloque de texto, y también se establece en el objeto `ElementFormat`. Funciona tanto con fuentes de dispositivo como incorporadas. FTE admite dos propiedades de espaciado manual, `trackingLeft`, que añade o retira píxeles por el lado izquierdo de un carácter, y `trackingRight`, que los añade o retira por el lado derecho. Cuando se utiliza el kerning (espaciado entre caracteres), el valor de espaciado manual se añade o se resta de los valores de kerning para cada par de caracteres.



***A.** Kerning.OFF **B.** TrackingRight=5, Kerning.OFF **C.** TrackingRight=-5, Kerning.OFF **D.** Kerning.ON **E.** TrackingRight=-5, Kerning.ON **F.** TrackingRight=-5, Kerning.ON*

```
var ef1:ElementFormat = new ElementFormat();
ef1.kerning = Kerning.OFF;

var ef2:ElementFormat = new ElementFormat();
ef2.kerning = Kerning.ON;
ef2.trackingLeft = 0.8;
ef2.trackingRight = 0.8;

var ef3:ElementFormat = new ElementFormat();
ef3.trackingRight = -0.2;
```

Saltos de línea para texto ajustado

La propiedad `breakOpportunity` del objeto `ElementFormat` determina qué caracteres se pueden usar para realizar el salto de línea cuando el texto ajustado ocupa varias líneas. La configuración predeterminada, `BreakOpportunity.AUTO`, utiliza propiedades estándar de Unicode, como por ejemplo dividir las líneas entre palabras o cuando hay guiones. Cuando la configuración es `BreakOpportunity.ALL` cualquier carácter se puede tratar como oportunidad de salto de línea, lo que resulta útil para crear efectos como texto distribuido a lo largo de un trazado.

```
var ef:ElementFormat = new ElementFormat();
ef.breakOpportunity = BreakOpportunity.ALL;
```

Tabulaciones

Para establecer tabulaciones en un bloque de texto, dichas tabulaciones se definen creando instancias de la clase TabStop. Los parámetros pasados al constructor `TabStop()` especifican cómo se alinea el texto con respecto a la tabulación. Estos parámetros especifican la posición de la tabulación y, para la alineación decimal, el valor respecto al que realizarla, expresado como una cadena. Normalmente, este valor es un punto decimal, pero también podría ser una coma, un símbolo de dólar o el símbolo del yen o el euro, por ejemplo. La línea de código siguiente crea una tabulación denominada tab1.

```
var tab1:TabStop = new TabStop(TabAlignment.DECIMAL, 50, ".");
```

Una vez creadas las tabulaciones para un bloque de texto, se asignan a la propiedad `tabStops` de una instancia de TextBlock. Dado que la propiedad `tabStops` requiere un vector, primero debe crearse un vector y después añadirle las tabulaciones. El vector permite asignarle un juego de tabulaciones al bloque de texto. El código siguiente crea una instancia de `Vector<TabStop>` y le añade un conjunto de objetos TabStop. Después, asigna las tabulaciones a la propiedad `tabStops` de una instancia de TextBlock.

```
var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
tabStops.push(tab1, tab2, tab3, tab4);
textBlock.tabStops = tabStops
```

Para obtener más información sobre objetos Vectors, consulte “Trabajo con conjuntos” en la página 159.

El ejemplo siguiente muestra los efectos que se obtienen con las distintas opciones de alineación de TabStop.

```
package {

    import flash.text.engine.*;
    import flash.display.Sprite;

    public class TabStopExample extends Sprite
    {
        public function TabStopExample()
        {
            var format:ElementFormat = new ElementFormat();
            format.fontDescription = new FontDdscription("Arial");
            format.fontSize = 16;

            var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
            tabStops.push(
                new TabStop(TabAlignment.START, 20),
                new TabStop(TabAlignment.CENTER, 140),
                new TabStop(TabAlignment.DECIMAL, 260, "."),
                new TabStop(TabAlignment.END, 380));
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = new TextElement(
                "\tt1\tt2\tt3\tt4\n" +
                "\tThis line aligns on 1st tab\n" +
                "\t\t\t\tThis is the end\n" +
                "\tThe following fragment centers on the 2nd tab:\t\t\n" +
                "\t\tit's on me\t\t\n" +
                "\tThe following amounts align on the decimal point:\n" +
                "\t\t\t45.00\t\n" +
                "\t\t\t75,320.00\t\n" +
                "\t\t\t6,950.00\t\n" +
                "\t\t\t7.01\t\n", format);

            textBlock.tabStops = tabStops;
            var yPosition:Number = 60;
            var previousTextLine:TextLine = null;
            var textLine:TextLine;
            var i:int;
            for (i = 0; i < 10; i++) {
                textLine = textBlock.createTextLine(previousTextLine, 1000, 0);
                textLine.x = 20;
                textLine.y = yPosition;
                addChild(textLine);
                yPosition += 25;
                previousTextLine = textLine;
            }
        }
    }
}
```

Ejemplo de FTE - News Layout

Este ejemplo de programación muestra el uso de Flash Text Engine con el diseño una sencilla página de periódico. La página incluye un titular grande, un subtítulo y una sección de cuerpo de varias columnas.

En primer lugar, cree un nuevo archivo FLA y añada el siguiente código al fotograma #2 de la capa predeterminada:

```
import com.example.programmingas3.newslayout.StoryLayout ;
// frame sc ript - create  a 3-columned arti cle layout
var story:StoryLayout = new StoryLayout(720, 500, 3, 10);
story.x = 20;
story.y = 80;
addChild(story);
stop();
```

StoryLayout.as es el script del controlador de este ejemplo. Establece el contenido, lee la información de estilo de una hoja de estilos externa y asigna estos estilos a objetos ElementFormat. Posteriormente crea el titular, el subtítulo y los elementos de texto de varias columnas.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.StyleSheet;
    import flash.text.engine.*;

    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.net.URLLoader;
    import flash.display.Sprite;
    import flash.display.Graphics;

    public class StoryLayout extends Sprite
    {
        public var headlineTxt:HeadlineTextField;
        public var subtitleTxt:HeadlineTextField;
        public var storyTxt:MultiColumnText;
        public var sheet:StyleSheet;
        public var h1_ElFormat:ElementFormat;
        public var h2_ElFormat:ElementFormat;
        public var p_ElFormat:ElementFormat;

        private var loader:URLLoader;

        public var paddingLeft:Number;
        public var paddingRight:Number;
        public var paddingTop:Number;
        public var paddingBottom:Number;

        public var preferredWidth:Number;
        public var preferredHeight:Number;

        public var numColumns:int;

        public var bgColor:Number = 0xFFFFFF;

        public var headline:String = "News Layout Example";
        public var subtitle:String = "This example formats text like a newspaper page using the
Flash Text Engine API. ";

        public var rawTestData:String =
        "From the part Mr. Burke took in the American Revolution, it was natural that I should
consider him a friend to mankind; and as our acquaintance commenced on that ground, it would
have been more agreeable to me to have had cause to continue in that opinion than to change it. " +
        "At the time Mr. Burke made his violent speech last winter in the English Parliament
```

```
against the French Revolution and the National Assembly, I was in Paris, and had written to him
but a short time before to inform him how prosperously matters were going on. Soon after this I
saw his advertisement of the Pamphlet he intended to publish: As the attack was to be made in a
language but little studied, and less understood in France, and as everything suffers by
translation, I promised some of the friends of the Revolution in that country that whenever Mr.
Burke's Pamphlet came forth, I would answer it. This appeared to me the more necessary to be
done, when I saw the flagrant misrepresentations which Mr. Burke's Pamphlet contains; and that
while it is an outrageous abuse on the French Revolution, and the principles of Liberty, it is
an imposition on the rest of the world. " +
        "I am the more astonished and disappointed at this conduct in Mr. Burke, as (from the
circumstances I am going to mention) I had formed other expectations. " +
        "I had seen enough of the miseries of war, to wish it might never more have existence
in the world, and that some other mode might be found out to settle the differences that should
occasionally arise in the neighbourhood of nations. This certainly might be done if Courts were
disposed to set honesty about it, or if countries were enlightened enough not to be made the
dupes of Courts. The people of America had been bred up in the same prejudices against France,
which at that time characterised the people of England; but experience and an acquaintance with
the French Nation have most effectually shown to the Americans the falsehood of those prejudices;
and I do not believe that a more cordial and confidential intercourse exists between any two
countries than between America and France. ";

        public function StoryLayout(w:int = 400, h:int = 200, cols:int = 3, padding:int =
10):void
        {
            this.preferredWidth = w;
            this.preferredHeight = h;

            this.numColumns = cols;

            this.paddingLeft = padding;
            this.paddingRight = padding;
            this.paddingTop = padding;
            this.paddingBottom = padding;

            var req:URLRequest = new URLRequest("story.css");
            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            this.sheet = new StyleSheet();
            this.sheet.parseCSS(loader.data);

            // convert headline styles to ElementFormat objects
            h1_ElFormat = getElFormat("h1", this.sheet);
            h1_ElFormat.typographicCase = TypographicCase.UPPERCASE;
            h2_ElFormat = getElFormat("h2", this.sheet);
            p_ElFormat = getElFormat("p", this.sheet);
            displayText();
        }

        public function drawBackground():void
        {
            var h:Number = this.storyTxt.y + this.storyTxt.height +
                           this.paddingTop + this.paddingBottom;
```

```
    var g:Graphics = this.graphics;
    g.beginFill(this.bgColor);
    g.drawRect(0, 0, this.width + this.paddingRight + this.paddingLeft, h);
    g.endFill();
}

/**
 * Reads a set of style properties for a named style and then creates
 * a TextFormat object that uses the same properties.
 */
public function getElFormat(styleName:String, ss:StyleSheet):ElementFormat
{
var style:Object = ss.getStyle(styleName);
if (style != null)
{
var colorStr:String = style.color;
if (colorStr != null && colorStr.indexOf("#") == 0)
{
    style.color = colorStr.substr(1);
}
        var fd:FontDescription = new FontDescription(
                            style.fontFamily,
                            style.fontWeight,
                            FontPosture.NORMAL,
                            FontLookup.DEVICE,
                            RenderingMode.NORMAL,
                            CFFHinting.NONE);
var format:ElementFormat = new ElementFormat(fd,
                        style.fontSize,
                        style.color,
                        1,
                        TextRotation.AUTO,
                        TextBaseline.ROMAN,
                        TextBaseline.USE_DOMINANT_BASELINE,
                        0.0,
                        Kerning.ON,
                        0.0,
                        0.0,
                        "en",
                                BreakOpportunity.AUTO,
                                DigitCase.DEFAULT,
                                DigitWidth.DEFAULT,
                        LigatureLevel.NONE,
                                TypographicCase.DEFAULT);

if (style.hasOwnProperty("letterSpacing"))
{
    format.trackingRight = style.letterSpacing;
}
}
return format;
}

public function displayText():void
{
headlineTxt = new HeadlineTextField(h1_ElFormat,headline,this.preferredWidth);
headlineTxt.x = this.paddingLeft;
```

```
        headlineTxt.y = 40 + this.paddingTop;
            headlineTxt.fitText(1);
            this.addChild(headlineTxt);

        subtitleTxt = new HeadlineTextField(h2_ElFormat,subtitle,this.preferredWidth);
        subtitleTxt.x = this.paddingLeft;
        subtitleTxt.y = headlineTxt.y + headlineTxt.height;
            subtitleTxt.fitText(2);
        this.addChild(subtitleTxt);

        storyTxt = new MultiColumnText(rawTestData, this.numColumns,
                        20, this.preferredWidth, this.preferredHeight, p_ElFormat);
        storyTxt.x = this.paddingLeft;
        storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
        this.addChild(storyTxt);

            drawBackground();
        }
    }
}
```

FormattedTextBlock.as se emplea como clase base para la creación de bloques de texto. También incluye funciones de utilidad para cambiar el tamaño de fuente y las mayúsculas o minúsculas.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class FormattedTextBlock extends Sprite
    {
        public var tb:TextBlock;
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var textWidth:int;
        public var totalTextLines:int;
        public var blockText:String;
        public var leading:Number = 1.25;
        public var preferredWidth:Number = 720;
        public var preferredHeight:Number = 100;

        public function FormattedTextBlock(ef:ElementFormat,txt:String, colW:int = 0)
        {
            this.textWidth = (colW==0) ? preferredWidth : colW;
            blockText = txt;
            ef1 = ef;
            tb = new TextBlock();
            tb.textJustifier = new SpaceJustifier("en",LineJustification.UNJUSTIFIED,false);
            te = new TextElement(blockText,this.ef1);
            tb.content = te;
            this.breakLines();
        }

        private function breakLines()
        {
            var textLine:TextLine = null;
            var y:Number = 0;
```

```
            var lineNum:int = 0;
            while (textLine = tb.createTextLine(textLine,this.textWidth,0,true))
            {
                textLine.x = 0;
                textLine.y = y;
                y += this.leading*textLine.height;
                this.addChild(textLine);
            }
            for (var i:int = 0; i < this.numChildren; i++)
            {
                TextLine(this.getChildAt(i)).validity = TextLineValidity.STATIC;
            }
            this.totalTextLines = this.numChildren;
        }

        private function rebreakLines()
        {
            this.clearLines();
            this.breakLines();
        }

        private function clearLines()
        {
            while(this.numChildren)
            {
                this.removeChildAt(0);
            }
        }

        public function changeSize(size:uint=12):void
        {
            if (size > 5)
            {
                var ef2:ElementFormat = ef1.clone();
                ef2.fontSize = size;
                te.elementFormat = ef2;
                this.rebreakLines();
            }
        }

        public function changeCase(newCase:String = "default"):void
        {
            var ef2:ElementFormat = ef1.clone();
            ef2.typographicCase = newCase;
            te.elementFormat = ef2;
        }
    }
}
```

HeadlineTextBlock.as amplía la clase FormattedTextBlock y se utiliza para crear titulares. Incluye una función para ajustar texto con un espacio definido en la página.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    public class HeadlineTextField extends FormattedTextBlock
    {

        public static var MIN_POINT_SIZE:uint = 6;
        public static var MAX_POINT_SIZE:uint = 128;

        public function HeadlineTextField(te:ElementFormat,txt:String,colW:int = 0)
        {
            super(te,txt);
        }

        public function fitText(maxLines:uint = 1, targetWidth:Number = -1):uint
        {
            if (targetWidth == -1)
            {
                targetWidth = this.width;
            }

            var pixelsPerChar:Number = targetWidth / this.blockText.length;
            var pointSize:Number = Math.min(MAX_POINT_SIZE,
                           Math.round(pixelsPerChar * 1.8 * maxLines));

            if (pointSize < 6)
            {
                // the point size is too small
                return pointSize;
            }

            this.changeSize(pointSize);
            if (this.totalTextLines > maxLines)
            {
                return shrinkText(--pointSize, maxLines);
            }
            else
            {
                return growText(pointSize, maxLines);
            }
        }

        public function growText(pointSize:Number, maxLines:uint = 1):Number
        {
            if (pointSize >= MAX_POINT_SIZE)
            {
                return pointSize;
            }

            this.changeSize(pointSize + 1);
            if (this.totalTextLines > maxLines)
            {
                // set it back to the last size
                this.changeSize(pointSize);
                return pointSize;
            }
            else
```

```
            {
                return growText(pointSize + 1, maxLines);
            }
        }

        public function shrinkText(pointSize:Number, maxLines:uint=1):Number
        {
            if (pointSize <= MIN_POINT_SIZE)
            {
                return pointSize;
            }
            this.changeSize(pointSize);

            if (this.totalTextLines > maxLines)
            {
                return shrinkText(pointSize - 1, maxLines);
            }
            else
            {
                return pointSize;
            }
        }
    }
}
```

MultiColumnText.as gestiona el formato del texto con un diseño de varias columnas. Muestra el uso flexible de un objeto TextBlock como base para crear, dar formato y situar líneas de texto.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.engine.*;

    public class MultiColumnText extends Sprite
    {
        private var tb:TextBlock;
        private var te:TextElement;
        private var numColumns:uint = 2;
        private var gutter:uint = 10;
        private var leading:Number = 1.25;
        private var preferredWidth:Number = 400;
        private var preferredHeight:Number = 100;
        private var colWidth:int = 200;

        public function MultiColumnText(txt:String = "",cols:uint = 2,
            gutter:uint = 10, w:Number = 400, h:Number = 100,
            ef:ElementFormat = null):void
        {
            this.numColumns = Math.max(1, cols);
            this.gutter = Math.max(1, gutter);

            this.preferredWidth = w;
            this.preferredHeight = h;

            this.setColumnWidth();

            var field:FormattedTextBlock = new FormattedTextBlock(ef,txt,this.colWidth);
```

```
            var totLines:int = field.totalTextLines;
            field = null;
            var linesPerCol:int = Math.ceil(totLines/cols);

            tb = new TextBlock();
            te = new TextElement(txt,ef);
            tb.content = te;
            var textLine:TextLine = null;
            var x:Number = 0;
            var y:Number = 0;
            var i:int = 0;
            var j:int = 0;
            while (textLine = tb.createTextLine(textLine,this.colWidth,0,true))
            {
                textLine.x = Math.floor(i/(linesPerCol+1))*(this.colWidth+this.gutter);
                textLine.y = y;
                y += this.leading*textLine.height;
                j++;
                if(j>linesPerCol)
                {
                    y = 0;
                    j = 0;
                }
                i++;

                this.addChild(textLine);
            }
        }

        private function setColumnWidth():void
        {
        this.colWidth = Math.floor( (this.preferredWidth -
            ((this.numColumns - 1) * this.gutter)) / this.numColumns);
        }

    }
}
```

Capítulo 22: Trabajo con mapas de bits

Además de sus funciones de dibujo vectorial, ActionScript 3.0 cuenta con la capacidad de crear imágenes de mapas de bits y de manipular los datos de píxeles de imágenes de mapas de bits externas cargadas en un archivo SWF. Gracias a la capacidad de acceder a los valores individuales de los píxeles y modificarlos, es posible crear efectos de imagen similares a los de los filtros y utilizar las funciones de ruido incorporadas para crear texturas y ruido aleatorio. En este capítulo se describen todas estas técnicas.

Fundamentos de la utilización de mapas de bits

Introducción a la utilización de mapas de bits

Cuando se trabaja con imágenes digitales, lo habitual es encontrar dos tipos principales de gráficos: de mapas de bits y vectoriales. Los gráficos de mapas de bits, también conocidos como gráficos raster, se componen de cuadrados diminutos (píxeles) distribuidos en una cuadrícula rectangular. Los gráficos vectoriales se componen de formas geométricas generadas matemáticamente, como líneas, curvas y polígonos.

Las imágenes de mapa de bits se definen mediante la altura y la anchura de la imagen, medidas en píxeles, y el número de bits que contiene cada píxel, que representa el número de colores que un píxel puede mostrar. En el caso de las imágenes de mapa de bits que utilizan el modelo de color RGB, los píxeles se componen de tres bytes: rojo, verde y azul. Cada uno de ellos contiene un valor entre 0 y 255. Cuando los bytes se combinan en el píxel, producen un color de un modo similar a un artista cuando mezcla pinturas de distintos colores. Por ejemplo, un píxel cuyos bytes tuvieran los siguientes valores, rojo - 255, verde - 102 y azul - 0, mostraría un color anaranjado vivo.

La calidad de una imagen de mapa de bits viene dada por la combinación de su resolución y su número de bits de profundidad del color. La *resolución* hace referencia al número de píxeles que contiene una imagen. A mayor número de píxeles, mayor resolución y mayor nitidez de la imagen. La *profundidad del color* indica la cantidad de información que puede contener un píxel. Por ejemplo, una imagen con una profundidad de color de 16 bits por píxel no puede representar el mismo número de colores que una imagen con una profundidad de color de 48 bits. De este modo, la imagen de 48 bits tendrá unas variaciones de tonalidad más suaves que su homóloga de 16 bits.

Dado que los gráficos de mapa de bits dependen de la resolución, pueden presentar problemas al ajustar su escala. Esto resulta evidente cuando se intentan ajustar la escala para aumentar su tamaño. Al aumentar de tamaño un mapa de bits se suelen perder detalles y calidad.

Formatos de mapa de bits

Las imágenes de mapa de bits se agrupan en una serie de formatos de archivo comunes. Estos formatos utilizan distintos tipos de algoritmos de compresión para reducir el tamaño del archivo y optimizar su calidad basándose en el uso que se vaya a dar a la imagen. Los formatos de imágenes de mapa de bits admitidos en Adobe Flash Player y Adobe AIR son BMP, GIF, JPG, PNG y TIFF.

BMP

El formato BMP (mapa de bits) es un formato de imagen predeterminado utilizado por el sistema operativo Microsoft Windows. No utiliza ningún algoritmo de compresión, por lo que el tamaño de los archivos suele ser grande.

GIF

El formato GIF (Graphics Interchange Format) fue desarrollado originalmente por CompuServe en 1987 como medio para transmitir imágenes con 256 colores (color de 8 bits). Este formato da lugar a archivos de pequeño tamaño y es ideal para imágenes que se van a usar en la Web. A causa de lo limitado de la paleta de colores de este formato, las imágenes GIF normalmente no son adecuadas para fotografías, que suelen requerir mayores variaciones de tonalidad y degradados de color. En las imágenes GIF es posible utilizar transparencia de un bit, lo que permite designar un color como vacío (o transparente). Esto hace que el color de fondo de una página Web se pueda ver a través de la imagen en la que se ha asignado la transparencia.

JPEG

Desarrollado por el Joint Photographic Experts Group (JPEG), el formato de imagen JPEG (a veces escrito JPG) utiliza un algoritmo de compresión con pérdida que permite combinar una profundidad de color de 24 bits con un tamaño de archivo reducido. En la compresión con pérdida, la imagen pierde calidad y datos cada vez que se guarda, pero se logra un tamaño de archivo menor. El formato JPEG resulta perfecto para fotografías, ya que puede mostrar millones de colores. La capacidad de controlar el grado de compresión aplicado a una imagen permite jugar con la calidad de la imagen y el tamaño del archivo.

PNG

El formato PNG (Portable Network Graphics) se originó como alternativa de código abierto al formato de archivo GIF, que está patentado. Los archivos PNG permiten una profundidad de color de hasta 64 bits, lo que equivale a más de 16 millones de colores. Dado que el formato PNG es relativamente nuevo, algunos navegadores antiguos no admiten este tipo de archivos. Al contrario que los JPG, los archivos PNG utilizan una compresión sin pérdida, lo que quiere decir que cuando la imagen se guarda no se pierden datos. Los archivos PNG también permiten el uso de transparencias alfa, con un máximo de 256 niveles de transparencia.

TIFF

El formato TIFF (formato de archivo de imagen etiquetada) era el formato multiplataforma más utilizado antes de la aparición del formato PNG. El inconveniente del formato TIFF proviene de las muchas variedades de TIFF existentes y de la ausencia de un único lector capaz de gestionarlas todas. Además, ningún navegador Web admite este formato actualmente. El formato TIFF puede utilizar compresión con o sin pérdidas y es capaz de gestionar espacios de color específicos del dispositivo (por ejemplo, CMYK).

Mapas de bits transparentes y opacos

Las imágenes de mapa de bits que utilizan los formatos GIF o PNG pueden tener un byte extra (canal alfa) para cada píxel, que representa su valor de transparencia.

Las imágenes GIF permiten una transparencia de un solo bit, lo que quiere decir que se puede especificar que un único color, de la paleta de 256, puede ser transparente. Por su parte, las imágenes PNG pueden tener hasta 256 niveles de transparencia. Esta función resulta especialmente útil cuando es necesario que las imágenes o el texto se fundan con los fondos.

ActionScript 3.0 replica este byte de transparencia extra de los píxeles dentro de la clase BitmapData. De manera similar al modelo de transparencia de PNG, la constante `BitmapDataChannel.ALPHA` ofrece hasta 256 niveles de transparencia.

Tareas comunes para trabajar con mapas de bits

En la siguiente lista aparecen diversas tareas que suelen llevarse a cabo al trabajar con imágenes de mapa de bits en ActionScript:

- Visualización de mapas de bits en pantalla
- Lectura y definición de valores de color de los píxeles
- Copia de datos de mapas de bits:
  - Creación de una copia exacta de un mapa de bits
  - Copia de datos de un canal de color de un mapa de bits en un canal de color de otro mapa de bits
  - Copia de una instantánea de un objeto de visualización de la pantalla en un mapa de bits
- Creación de ruido y texturas en imágenes de mapas de bits
- Desplazamiento por mapas de bits

Conceptos y términos importantes

La siguiente lista incluye términos importantes utilizados en este capítulo:

- Alfa: nivel de transparencia (o, para ser más precisos, opacidad) de un color o una imagen. La cantidad de alfa suele describirse como el valor del *canal alfa*.
- Color ARGB: esquema de color en el que el color de cada píxel es una mezcla de los valores de los colores rojo, verde y azul, y su transparencia se especifica como un valor alfa.
- Canal de color: normalmente, los colores se representan como una mezcla de varios colores básicos (en el campo de los gráficos digitales suelen ser rojo, verde y azul). Cada color básico se considera un canal de color. La mezcla de la cantidad de color de cada canal determina el color final.
- Profundidad de color: también denominada *profundidad de bits*; este concepto hace referencia a la cantidad de memoria del equipo dedicada a cada píxel que, a su vez, determina el número de posibles colores que se pueden representar en la imagen.
- Píxel: unidad de información mínima de una imagen de mapa de bits (esencialmente, un punto de color).
- Resolución: dimensiones en píxeles de una imagen, que determinan el nivel de detalle que contiene la imagen. La resolución se suele expresar en términos del número de píxeles de anchura y de altura.
- Color RGB: esquema de color en el que el color de cada píxel se representa como una mezcla de los valores de los colores rojo, verde y azul.

Ejecución de los ejemplos del capítulo

A medida que progrese en el estudio de este capítulo, podría desear probar el código de ejemplo. Como este capítulo se centra en la creación y manipulación de contenido visual, para probar el código hay que ejecutarlo y ver los resultados en el archivo SWF creado.

Para probar los ejemplos de código de este capítulo:

1. Cree un documento vacío utilizando la herramienta de edición de Flash.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el código en el panel Script.
4. Ejecute el programa utilizando Control > Probar película.

Puede ver el resultado del código en el archivo SWF creado.

Casi todos los ejemplos incluyen código que crea una imagen de mapa de bits, por lo que puede probar el código directamente sin necesidad de suministrar contenido de mapa de bits. Como alternativa, si desea probar el código con una de sus imágenes, puede importar la imagen en Adobe Flash CS4 Professional o cargar la imagen externa en el archivo SWF de prueba y utilizar los datos de mapa de bits con el código de ejemplo. Para obtener instrucciones sobre la carga de imágenes externas, consulte "Carga dinámica de contenido de visualización" en la página 320.

Las clases Bitmap y BitmapData

La clases principales de ActionScript 3.0 para trabajar con imágenes de mapa de bits son la Bitmap que se utiliza para visualizar imágenes de mapa de bits en pantalla, y la clase BitmapData, que se utiliza para acceder a datos de imagen sin procesar de un mapa de bits y manipularlos.

Aspectos básicos de la clase Bitmap

Como subclase de la clase DisplayObject, la clase Bitmap es la clase principal de ActionScript 3.0 para mostrar imágenes de mapa de bits. Estas imágenes se pueden cargar en Flash Player o Adobe AIR mediante la clase flash.display.Loader o se pueden crear dinámicamente utilizando el constructor `Bitmap()`. Al cargar una imagen de una fuente externa, los objetos Bitmap sólo pueden usar los formatos de imagen GIF, JPEG o PNG. Una vez creada, la instancia de Bitmap se puede considerar como un envolvente de un objeto BitmapData que se debe representar en el escenario. Dado que las instancias de Bitmap son objetos de visualización, también se pueden utilizar todas las características y funcionalidades de los objetos de visualización para manipularlas. Para obtener más información sobre la utilización de los objetos de visualización, consulte "Programación de la visualización" en la página 278.

Ajuste a píxeles y suavizado

Además de las funcionalidades comunes a todos los objetos de visualización, la clase Bitmap proporciona algunas características adicionales específicas de las imágenes de mapa de bits.

La propiedad `pixelSnapping` es similar a la característica de ajustar a píxeles de la herramienta de edición Flash, y determina si un objeto Bitmap se ajusta a su píxel más cercano o no. Esta propiedad acepta una de las tres constantes definidas en la clase PixelSnapping: `ALWAYS`, `AUTO` y `NEVER`.

La sintaxis para aplicar el ajuste a píxeles es la siguiente:

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

Suele ocurrir que, cuando se escalan las imágenes de mapa de bits, éstas se vuelven difusas y distorsionadas. Para reducir esta distorsión se puede utilizar la propiedad `smoothing` de la clase BitmapData. Cuando esta propiedad booleana está definida como `true`, al escalar la imagen, los píxeles de ésta se suavizan. Esto otorga a la imagen una apariencia más clara y natural.

Aspectos básicos de la clase BitmapData

La clase BitmapData, que pertenece al paquete flash.display, se asemeja a una instantánea fotográfica de los píxeles que contiene una imagen de mapa de bits cargada o creada dinámicamente. Esta instantánea se representa mediante un conjunto que contiene los datos de los píxeles del objeto. La clase BitmapData también contiene una serie de métodos incorporados que resultan muy útiles a la hora de crear y manipular los datos de los píxeles.

Se puede usar el siguiente código para crear una instancia de un objeto BitmapData:

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number, transparent:Boolean,
fillColor:uinit);
```

Los parámetros `width` y `height` especifican el tamaño del mapa de bits. El valor máximo para cualquiera de ellos es de 2880 píxeles. El parámetro `transparent` especifica si entre los datos del mapa de bits se incluye un canal alfa (`true`) o no (`false`). El parámetro `fillColor` es un valor de color de 32 bits que especifica el color de fondo, así como el valor de la transparencia (si se le ha definido como `true`). En el siguiente ejemplo se crea un objeto BitmapData con un fondo anaranjado y una transparencia del 50 por ciento:

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

Para representar en la pantalla un objeto BitmapData recién creado, se debe asignar a una instancia de Bitmap o envolverse en ella. Para ello, se puede pasar el objeto BitmapData como parámetro del constructor del objeto Bitmap o asignarlo a la propiedad `bitmapData` de una instancia de Bitmap existente. También es necesario añadir la instancia de Bitmap a la lista de visualización llamando para ello a los métodos `addChild()` o `addChildAt()` del contenedor del objeto de visualización que contendrá a la instancia de Bitmap. Para obtener más información sobre el trabajo con la lista de visualización, consulte "Añadir objetos de visualización a la lista de visualización" en la página 286.

En el siguiente ejemplo se crea un objeto BitmapData con un relleno rojo y se muestra en una instancia de Bitmap:

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

Manipulación de píxeles

La clase BitmapData contiene un conjunto de métodos que permiten manipular los valores de los datos de los píxeles.

Manipulación de píxeles individuales

Cuando se desea cambiar la apariencia de una imagen de mapa de bits a nivel de sus píxeles, en primer lugar es necesario obtener los valores de color de los píxeles que contiene el área que se pretende manipular. Para leer esos valores de los píxeles se utiliza el método `getPixel()`.

El método `getPixel()` obtiene el valor RGB del par de coordenadas x, y (píxel) que se le pasan como parámetros. Si alguno de los píxeles que se desea manipular incluye información de transparencia (canal alfa), será necesario emplear el método `getPixel32()`. Este método también lee un valor RGB pero, al contrario que `getPixel()`, el valor devuelto por `getPixel32()` contiene datos adicionales que representan el valor del canal alfa (transparencia) del píxel seleccionado.

Por otra parte, si simplemente se desea cambiar el color o la transparencia de un píxel que pertenece a un mapa de bits, se pueden usar los métodos `setPixel()` o `setPixel32()`. Para definir el color de un píxel basta con pasar las coordenadas x, y, además del valor del color, a uno de estos métodos.

En el siguiente ejemplo se utiliza `setPixel()` para dibujar una cruz en un fondo BitmapData verde. A continuación, se emplea `getPixel()` para leer el valor del color del píxel que se encuentra en las coordenadas 50, 50 y se realiza un seguimiento del valor devuelto.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
    var red:uint = 0xFF0000;
    myBitmapData.setPixel(50, i, red);
    myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);

var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

Si se desea leer el valor de un grupo de píxeles, en lugar del de uno solo, se debe usar el método `getPixels()`. Este método genera un conjunto de bytes a partir de una región rectangular de datos de píxeles que se transmite como parámetro. Cada uno de los elementos del conjunto de bytes (dicho de otro modo, los valores de los píxeles) es un entero sin signo, es decir, un valor de píxel no multiplicado de 32 bits.

A la inversa, para cambiar (o definir) el valor de un grupo de píxeles, se usa el método `setPixels()`. Este método recibe dos parámetros (`rect` y `inputByteArray`), que se combinan para dar lugar a una región rectangular (`rect`) de datos de píxeles (`inputByteArray`).

A medida que se leen (y escriben) los datos de `inputByteArray`, se llama al método `ByteArray.readUnsignedInt()` para cada uno de los píxeles del conjunto. Si, por algún motivo, `inputByteArray` no contiene todo un rectángulo de datos de píxeles, el método deja de procesar los datos de imagen en ese punto.

Es importante recordar que, tanto para leer como para definir los datos de los píxeles, el conjunto de bytes espera valores de píxeles de 32 bits compuestos por los canales alfa, rojo, verde y azul (ARGB).

En el siguiente ejemplo se utilizan los métodos `getPixels()` y `setPixels()` para copiar un grupo de píxeles de un objeto BitmapData a otro:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false, 0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false, 0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

Detección de colisiones a nivel de píxeles

El método `BitmapData.hitTest()` lleva a cabo una detección de colisiones a nivel de píxeles entre los datos de un mapa de bits y otro objeto o punto.

El método `BitmapData.hitTest()` acepta cinco parámetros:

- `firstPoint` (Point): este parámetro hace referencia a la posición del píxel de la esquina superior izquierda del primer objeto BitmapData sobre el que se realizará la comprobación de colisiones.
- `firstAlphaThreshold` (uint): este parámetro especifica el valor de canal alfa más alto que se considera opaco para esta prueba.
- `secondObject` (objeto): este parámetro representa el área de impacto. El objeto `secondObject` puede ser un objeto Rectangle, Point, Bitmap o BitmapData. Este objeto representa el área de impacto sobre la que se realizará la detección de colisiones.
- `secondBitmapDataPoint` (Point): este parámetro opcional se utiliza para definir la posición de un píxel en el segundo objeto BitmapData. y sólo se utiliza cuando el valor de `secondObject` es un objeto BitmapData. El valor predeterminado es `null`.
- `secondAlphaThreshold` (uint): este parámetro opcional representa el valor de canal alfa más alto que se considera opaco en el segundo objeto BitmapData. El valor predeterminado es 1. Este parámetro sólo se utiliza cuando el valor de `secondObject` es un objeto BitmapData y los dos objetos BitmapData son transparentes.

Al llevar a cabo la detección de colisiones sobre imágenes opacas, es conveniente recordar que ActionScript trata la imagen como si fuera un rectángulo (o recuadro delimitador) totalmente opaco. Por otra parte, al realizar la prueba de impactos a nivel de píxeles en imágenes transparentes, es necesario que las dos imágenes sean transparentes. Además, ActionScript utiliza los parámetros de umbral alfa para determinar en qué punto los píxeles pasan de ser transparentes a opacos.

En el siguiente ejemplo se crean tres imágenes de mapa de bits y se realiza una detección de colisiones de píxeles utilizando dos puntos de colisión distintos (uno devuelve false y el otro true):

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;

// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

Copiar datos de mapas de bits

Para copiar datos del mapa de bits de una imagen a otra, puede utilizar varios métodos: `clone()`, `copyPixels()`, `copyChannel()` y `draw()`.

Como indica su nombre, el método `clone()` permite clonar datos de mapas de bits (o tomar muestras de ellos) de un objeto BitmapData a otro. Cuando se le llama, este método devuelve un nuevo objeto BitmapData que es una copia exacta de la instancia original que se ha clonado.

En el siguiente ejemplo se clona una copia de un cuadrado de color naranja (el elemento principal) y se coloca el clon junto al cuadro naranja principal:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false, 0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

El método `copyPixels()` constituye una forma rápida y sencilla de copiar píxeles de un objeto BitmapData a otro. Este método toma una instantánea rectangular (definida por el parámetro `sourceRect`) de la imagen de origen y la copia en un área rectangular distinta (del mismo tamaño). La ubicación del rectángulo recién "pegado" se define mediante el parámetro `destPoint`.

El método `copyChannel()` toma una muestra de un valor de canal de color predefinido (alfa, rojo, verde o azul) de un objeto BitmapData de origen y la copia en un canal de un objeto BitmapData de destino. Llamar a este método no afecta a los demás canales del objeto BitmapData de destino.

El método `draw()` dibuja, o representa, el contenido gráfico de un objeto Sprite, de un clip de película o de otro objeto de visualización de origen en un nuevo mapa de bits. Mediante los parámetros `matrix`, `colorTransform`, `blendMode` y `clipRect` de destino, es posible modificar la forma en la que se representa el nuevo mapa de bits. Este método utiliza el vector procesado en Flash Player y AIR para generar los datos.

Al llamar a `draw()`, se le pasa el objeto de origen (objeto Sprite, clip de película u otro objeto de visualización) como primer parámetro, tal y como puede verse a continuación:

```
myBitmap.draw(movieClip);
```

Si al objeto de origen se le ha aplicado alguna transformación (color, matriz, etc.) después de haber sido cargado originalmente, éstas no se copiarán en el nuevo objeto. Para copiar las transformaciones en el nuevo mapa de bits, es necesario copiar el valor de la propiedad `transform` del objeto original en la propiedad `transform` del objeto Bitmap utilizado por el nuevo objeto BitmapData.

Creación de texturas con funciones de ruido

Para modificar la apariencia de un mapa de bits, se le puede aplicar un efecto de ruido utilizando para ello los métodos `noise()` o `perlinNoise()`. Un efecto de ruido puede asemejarse a la estática que aparece en una pantalla de televisión no sintonizada.

Para aplicar un efecto de ruido a un mapa de bits se utiliza el método `noise()`. Este método aplica un valor de color aleatorio a los píxeles que se hallan dentro de un área especificada de una imagen de mapa de bits.

Este método acepta cinco parámetros:

- `randomSeed` (int): número de inicialización aleatorio que determinará el patrón. A pesar de lo que indica su nombre, este número crea los mismos resultados siempre que se pasa el mismo número. Para obtener un resultado realmente aleatorio, es necesario utilizar el método `Math.random()` a fin de pasar un número aleatorio a este parámetro.
- `low` (uint): este parámetro hace referencia al valor más bajo que se generará para cada píxel (de 0 a 255). El valor predeterminado es 0. Si se establece un valor más bajo, se originará un patrón de ruido más oscuro, mientras que con un valor más alto, el patrón será más brillante.
- `high` (uint): este parámetro hace referencia al valor más alto que se generará para cada píxel (de 0 a 255). El valor predeterminado es 255. Si se establece un valor más bajo, se originará un patrón de ruido más oscuro, mientras que con un valor más alto, el patrón será más brillante.
- `channelOptions` (uint): este parámetro especifica el canal de color del objeto de mapa de bits al que se aplicará el patrón de ruido. El número puede ser una combinación de cualquiera de los cuatro valores ARGB de los canales de color. El valor predeterminado es 7.
- `grayScale` (Boolean): cuando se define como `true`, este parámetro aplica el valor de `randomSeed` a los píxeles del mapa de bits, de modo que se elimina todo el color de la imagen. El canal alfa no se ve afectado por este parámetro. El valor predeterminado es `false`.

En el siguiente ejemplo se crea una imagen de mapa de bits y se le aplica un patrón de ruido azul:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
var image:Bitmap = new Bitmap(myBitmap);
addChild(image);
```

Si se desea crear una textura de apariencia más orgánica, es aconsejable utilizar el método `perlinNoise()`. El método `perlinNoise()` produce texturas orgánicas realistas que resultan idóneas para humo, nubes, agua, fuego e incluso explosiones.

Dado que se genera mediante un algoritmo, el método `perlinNoise()` utiliza menos memoria que las texturas basadas en mapas de bits. Sin embargo, puede repercutir en el uso del procesador y ralentizar el contenido creado por Flash, haciendo que la pantalla se vuelva a dibujar más lentamente que la velocidad de fotogramas, sobre todo en equipos antiguos. Principalmente esto se debe a los cálculos en punto flotante que se deben ejecutar para procesar los algoritmos de ruido Perlin.

Este método acepta nueve parámetros (los seis primeros son obligatorios):

- `baseX` (Number): determina el valor x (tamaño) de los patrones creados.
- `baseY` (Number): determina el valor y (tamaño) de los patrones creados.
- `numOctaves` (uint): número de octavas o funciones de ruido individuales que se van a combinar para crear este ruido. A mayor número de octavas, mayor detalle pero también más tiempo de procesamiento.
- `randomSeed` (int): el número de inicialización aleatorio funciona exactamente igual que en la función `noise()`. Para obtener un resultado realmente aleatorio, es necesario utilizar el método `Math.random()` para transmitir un número aleatorio a este parámetro.
- `stitch` (Boolean): si se establece en `true`, este método intenta unir (o suavizar) los bordes de transición de la imagen para crear texturas continuas que se pueden utilizar como mosaicos para rellenos de mapas de bits.
- `fractalNoise` (Boolean): este parámetro hace referencia a los bordes de los degradados generados mediante este método. Si se define como `true`, el método genera ruido fractal que suaviza los bordes del efecto. Si se define como `false`, genera turbulencia. Una imagen con turbulencia presenta discontinuidades visibles en el degradado que pueden producir efectos visuales más nítidos, como llamas u olas del mar.
- `channelOptions` (uint): el parámetro `channelOptions` funciona del mismo modo que en el método `noise()`. Con él es posible especificar a qué canal de color (del mapa de bits) se aplicará el patrón de ruido. El número puede ser una combinación de cualquiera de los cuatro valores ARGB de los canales de color. El valor predeterminado es 7.
- `grayScale` (Boolean): el parámetro `grayScale` funciona del mismo modo que en el método `noise()`. Si se define como `true`, aplica el valor de `randomSeed` a los píxeles del mapa de bits, de modo que se elimina todo el color de la imagen. El valor predeterminado es `false`.
- `offsets` (Conjunto): conjunto de puntos que corresponde a desplazamientos x e y para cada octava. Mediante la manipulación de los valores de desplazamiento se pueden mover suavemente las capas de la imagen. Cada punto de la conjunto de desplazamiento afecta a una función de ruido de octava específica. El valor predeterminado es `null`.

En el siguiente ejemplo se crea un objeto BitmapData de 150 x 150 píxeles que llama al método `perlinNoise()` para generar un efecto de nubes azul y verde:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0x00FF0000);

var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myBitmap);
```

Desplazarse por mapas de bits

Supongamos que se ha creado una aplicación de mapas de calles en la que cada vez que el usuario mueve el mapa, hay que actualizar la vista (incluso si el mapa sólo se ha movido unos pocos píxeles).

Una forma de crear esta funcionalidad sería volver a representar una nueva imagen que contuviese la vista actualizada del mapa cada vez que el usuario lo moviese. También se podría crear una única imagen de gran tamaño y utilizar el método `scroll()`.

El método `scroll()` copia un mapa de bits que aparece en pantalla y lo pega en una nueva posición desplazada, que viene especificada por los parámetros (`x`, `y`). Si una parte del mapa de bits se encuentra fuera de la pantalla, se produce la impresión que la imagen se ha desplazado. Cuando se combina con una función de temporizador (o un evento `enterFrame`), se puede hacer que la imagen parezca moverse o desplazarse.

En el siguiente ejemplo se parte del ejemplo anterior del ruido de Perlin y se genera una imagen de mapa de bits mayor (tres cuartas partes de la cual se representan fuera de la pantalla). A continuación, se aplica el método `scroll()` junto con un detector de eventos `enterFrame` que desplaza la imagen un píxel en dirección diagonal descendente. Este método se llama cada vez que se entra en el fotograma y, de este modo, las partes de la imagen que quedan fuera de la pantalla se representan en el escenario a medida que la imagen se desplaza hacia abajo.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false, 0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);

function scrollBitmap(event:Event):void
{
    myBitmapDataObject.scroll(1, 1);
}
```

Aprovechamiento de la técnica de mipmapping

Los *mapas MIP* (también denominados *mipmaps*), son mapas de bits agrupados de forma conjunta y asociados a una textura para aumentar el rendimiento y la calidad de representación en tiempo de ejecución. Flash Player 9.0.115.0. y las versiones posteriores y AIR implementan esta tecnología (el proceso se denomina *mipmapping*), creando versiones optimizadas de escala variable de cada mapa de bits (comenzando en un 50%).

Flash Player y AIR crean mapas MIP para los mapas de bits (archivos JPEG, GIF o PNG) que se pueden visualizar con la clase Loader de ActionScript 3.0, con un mapa de bits de la biblioteca de la herramienta de edición de Flash o mediante un objeto BitmapData. Flash Player crea mapas MIP para los mapas de bits que se visualizan utilizando la función `loadMovie()` de ActionScript 2.0.

Los mapas MIP no se aplican a objetos filtrados o clips de película almacenados en caché de mapa de bits. Sin embargo, los mapas MIP se aplican si dispone de transformaciones de mapa de bits en un objeto de visualización filtrado, aunque el mapa de bits esté en un contenido con máscara.

El mipmapping con Flash Player y AIR se realiza automáticamente, pero puede seguir algunas directrices para garantizar que sus imágenes aprovechan esta optimización:

- Para la reproducción de vídeo, establezca la propiedad `smoothing` en `true` para el objeto Video (consulte la clase Video).
- Para los mapas de bits, la propiedad `smoothing` no tiene que establecerse en `true`, pero las mejoras de calidad serán más visibles si los mapas de bits utilizan el suavizado.
- Utilice tamaños de mapa de bits que sean divisibles por 4 u 8 para imágenes bidimensionales (por ejemplo, 640 x 128, que se puede reducir del siguiente modo: 320 x 64 > 160 x 32 > 80 x 16 > 40 x 8 > 20 x 4 > 10 x 2 > 5 x 1) y 2^n para texturas tridimensionales. Los mapas MIP se generan a partir de mapas de bits de una anchura y altura de 2^n (por ejemplo, 256 x 256, 512 x 512, 1024 x 1024). El mipmapping se detiene cuando Flash Player o AIR encuentran una altura o anchura distinta.

Ejemplo: Luna giratoria animada

Con el ejemplo de la luna giratoria animada se muestran estas técnicas para trabajar con objetos Bitmap y datos de imagen de mapa de bits (objetos BitmapData). El ejemplo crea una animación de una luna esférica giratoria utilizando una imagen plana de la superficie de la luna como datos de imagen sin procesar. Se muestran las siguientes técnicas:

- Carga de una imagen externa y acceso a sus datos de imagen sin procesar.
- Creación de una animación copiando píxeles de forma repetida de diferentes partes de una imagen de origen.
- Creación de una imagen de mapa de bits estableciendo valores de píxel.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación de la luna giratoria animada se encuentran en la carpeta Samples/SpinningMoon. La aplicación consta de los siguientes archivos:

Archivo	Descripción
SpinningMoon.mxml o SpinningMoon.fla	Archivo principal de la aplicación en Flex (MXML) o en Flash (FLA).
com/example/programmingas3/moon/MoonSphere.as	Clase que realiza la funcionalidad de cargar, visualizar y animar la luna.
moonMap.png	Archivo de imagen que contiene una fotografía de la superficie de la luna, que se carga y se utiliza para crear la luna giratoria animada.

Carga de una imagen externa como datos de mapa de bits

La primera tarea importante que se realiza en este ejemplo es cargar un archivo de imagen externo, que es una fotografía de la superficie de la luna. La operación de carga se administra mediante dos métodos en la clase MoonSphere: el constructor `MoonSphere()`, donde se inicia el proceso de carga y el método `imageLoadComplete()`, que se llama cuando la imagen externa está completamente cargada.

La carga de una imagen externa es similar a la carga de un archivo SWF externo; ambos utilizan una instancia de la clase flash.display.Loader para realizar la operación de carga. El código real del método `MoonSphere()` que inicia la carga de la imagen se presenta del siguiente modo:

```
var imageLoader:Loader = new Loader();
imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoadComplete);
imageLoader.load(new URLRequest("moonMap.png"));
```

La primera línea declara la instancia de Loader denominada `imageLoader`. La tercera línea inicia realmente el proceso de carga llamando al método `load()` del objeto Loader, transmitiendo una instancia de URLRequest que representa la URL de la imagen que se va a cargar. La segunda línea configura el detector de eventos que se activará cuando la imagen se haya cargado por completo. Se debe tener en cuenta que el método `addEventListener()` no se llama en la propia instancia de Loader; en cambio, se llama en la propiedad `contentLoaderInfo` del objeto Loader. La propia instancia de Loader no distribuye eventos relacionados con el contenido que se está cargando. No obstante, su propiedad `contentLoaderInfo` contiene una referencia al objeto LoaderInfo que se asocia al contenido que se está cargando en el objeto Loader (la imagen externa en este caso). El objeto LoaderInfo proporciona varios eventos relacionados con el curso y la finalización de la carga de contenido externo, incluyendo el evento `complete` (`Event.COMPLETE`) que activará una llamada al método `imageLoadComplete()` cuando la imagen se haya cargado por completo.

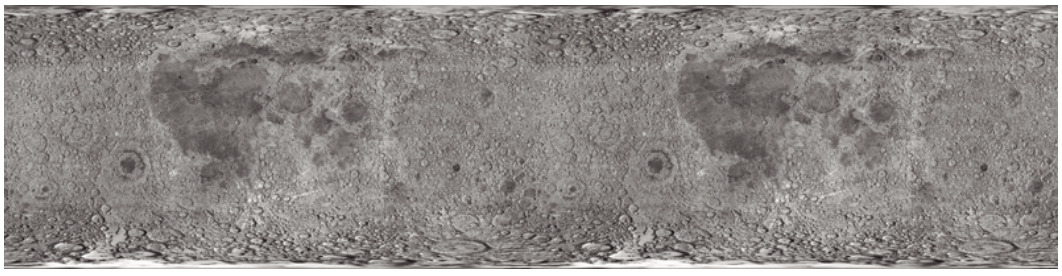
Aunque el inicio de la carga de la imagen externa es una parte importante del proceso, es igualmente relevante saber qué hacer cuando finalice la carga. Tal y como se muestra en el código anterior, la función `imageLoadComplete()` se llama cuando se carga la imagen. La función realiza varias operaciones con los datos de imagen cargados, lo que se describe en secciones posteriores. No obstante, para utilizar los datos de imagen, es necesario acceder a los mismos. Si se usa un objeto Loader para cargar una imagen externa, la imagen cargada se convierte en una instancia de Bitmap, que se asocia como objeto de visualización secundario del objeto Loader. En este caso, la instancia de Loader está disponible para el método del detector de eventos como parte del objeto de evento que se transmite al método como parámetro. Las primeras líneas del método `imageLoadComplete()` se presentan del siguiente modo:

```
private function imageLoadComplete(event:Event):void
{
    textureMap = event.target.content.bitmapData;
    ...
}
```

Observe que el parámetro del objeto de evento se denomina `event` y es una instancia de la clase Event. Todas las instancias de la clase Event disponen de una propiedad `target`, que hace referencia al objeto que activa el evento (en este caso, la instancia de LoaderInfo en la que se llamó el método `addEventListener()`, tal y como se describió anteriormente). Por su parte, el objeto LoaderInfo tiene una propiedad `content` que (una vez completado el proceso de carga) contiene la instancia de Bitmap con la imagen de mapa de bits cargada. Si desea visualizar la imagen directamente en pantalla, puede asociar esta instancia de Bitmap (`event.target.content`) a un contenedor de objetos de visualización. (También puede asociar el objeto Loader a un contenedor de objetos de visualización). No obstante, en este ejemplo el contenido cargado se utiliza como origen de los datos de la imagen sin procesar en lugar de mostrarse en pantalla. Por ello, la primera línea del método `imageLoadComplete()` lee la propiedad `bitmapData` de la instancia de Bitmap cargada (`event.target.content.bitmapData`) y la almacena en la variable de la instancia denominada `textureMap`, que, tal y como se describe en la siguiente sección, se utiliza como origen de los datos de imagen para crear la animación de la luna en rotación.

Creación de animación con copia de píxeles

Una definición básica de animación es la ilusión de movimiento o cambio, creado mediante el cambio de una imagen en el tiempo. En este ejemplo, el objetivo es crear la ilusión de una rotación de la luna esférica alrededor de su eje vertical. Sin embargo, para los objetivos de la animación, se puede omitir el aspecto de distorsión esférica del ejemplo. Observe la imagen real que se carga y se utiliza como el origen de los datos de imagen de la luna:



Como se puede ver, la imagen no es una o varias esferas; se trata de una fotografía rectangular de la superficie de la luna. Debido a que la foto fue tomada exactamente en el ecuador de la luna, las partes de la imagen más cercanas a la parte superior e inferior de la imagen están expandidas y distorsionadas. Para eliminar la distorsión de la imagen y hacerla parecer esférica, utilizaremos un filtro de mapa de desplazamiento, tal y como se describe más adelante. Son embargo. debido a que esta imagen de origen es un rectángulo, para crear la ilusión de que la esfera está girando, el código simplemente necesita deslizar la fotografía de la superficie de la luna horizontalmente, tal y como se indica en los siguientes párrafos.

Se debe tener en cuenta que la imagen incluye realmente dos copias de la fotografía de la superficie de la luna próximas entre sí. Esta imagen es la de origen a partir de la que los datos de imagen se copian repetidamente para crear la apariencia de movimiento. Al tener dos copias de la imagen próximas entre sí, un efecto de desplazamiento ininterrumpido y continuo puede ser más fácil de crear. Analicemos el proceso de animación paso a paso para ver su funcionamiento.

El proceso incluye dos objetos de ActionScript independientes. En primer lugar se dispone de la imagen de origen cargada, que en el código se representa mediante una instancia de BitmapData denominada `textureMap`. Tal y como se ha descrito anteriormente, `textureMap` se llena con datos de imagen una vez cargada la imagen externa, utilizando este código:

```
textureMap = event.target.content.bitmapData;
```

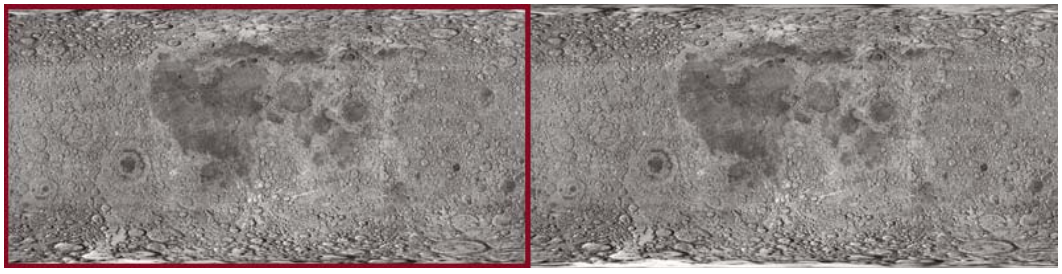
El contenido de `textureMap` es la imagen mostrada anteriormente. Asimismo, para crear la rotación animada, en el ejemplo se utiliza una instancia de Bitmap denominada `sphere`, que es el objeto de visualización real que muestra la imagen de la luna en pantalla. Al igual que sucede con `textureMap`, el objeto `sphere` se crea y se llena con sus datos de imagen inicial en el método `imageLoadComplete()`, empleando el siguiente código:

```
sphere = new Bitmap();
sphere.bitmapData = new BitmapData(textureMap.width / 2, textureMap.height);
sphere.bitmapData.copyPixels(textureMap,
                         new Rectangle(0, 0, sphere.width, sphere.height),
                         new Point(0, 0));
```

Tal y como muestra el código, se crea una instancia de `sphere`. Su propiedad `bitmapData` (los datos de la imagen sin procesar que se muestran mediante `sphere`) se crea con la misma altura y la mitad de anchura de `textureMap`. Es decir, el contenido de `sphere` será el tamaño de una fotografía de la luna (ya que la imagen `textureMap` contiene dos fotografías de la luna, una junto a otra). A continuación, la propiedad `bitmapData` se llena con los datos de imagen utilizando su método `copyPixels()`. Los parámetros en la llamada al método `copyPixels()` indican varios puntos:

- El primer parámetro indica que los datos de imagen se copian desde `textureMap`.
- El segundo parámetro, una nueva instancia de Rectangle, especifica desde qué parte de `textureMap` se debe tomar la instantánea de la imagen; en este caso, la instantánea es un rectángulo que comienza en la esquina superior izquierda de `textureMap` (indicado mediante los dos primeros parámetros `Rectangle(): 0, 0`) y la altura y anchura de la instantánea del rectángulo coinciden con las propiedades `width` y `height` de `sphere`.
- El tercer parámetro, una nueva instancia de Point con los valores x e y de `0`, define el destino de los datos de píxel; en este caso, la esquina superior izquierda de (0, 0) de `sphere.bitmapData`.

Representado visualmente, el código copia los píxeles desde `textureMap` destacado en la siguiente imagen y los pega en `sphere`. Es decir, el contenido de BitmapData de `sphere` es la parte de `textureMap` resaltada aquí:



No obstante, recuerde que esto sólo es el estado inicial de `sphere`; el contenido de la primera imagen que se copia en `sphere`.

Con la imagen de origen cargada y `sphere` creada, la tarea final realizada por el método `imageLoadComplete()` es configurar la animación. La animación se activa mediante una instancia de Timer denominada`rotationTimer`, que se crea y se inicia mediante el siguiente código:

```
var rotationTimer:Timer = new Timer(15);
rotationTimer.addEventListener(TimerEvent.TIMER, rotateMoon);
rotationTimer.start();
```

En primer lugar, el código crea la instancia de Timer denominada `rotationTimer`; el parámetro transmitido al constructor `Timer()` indica que `rotationTimer` debe activar su evento `timer` cada 15 millisegundos. A continuación, se llama al método `addEventListener()`, especificando que cuando sucede el evento `timer` (`TimerEvent.TIMER`), se llama al método `rotateMoon()`. Finalmente, timer se inicia realmente llamando a su método `start()`.

Debido al modo en que se define `rotationTimer`, aproximadamente cada 15 millisegundos Flash Player llama al método `rotateMoon()` en la clase MoonSphere, que es donde sucede la animación de la luna. El código fuente del método rotateMoon() se presenta del siguiente modo:

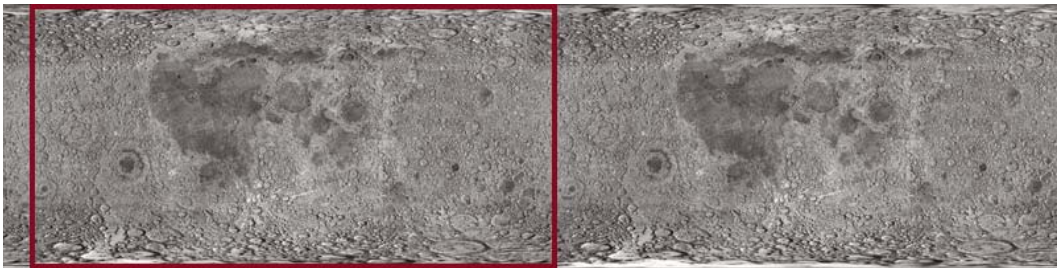
```
private function rotateMoon(event:TimerEvent):void
{
    sourceX += 1;
    if (sourceX > textureMap.width / 2)
    {
        sourceX = 0;
    }

    sphere.bitmapData.copyPixels(textureMap,
                                 new Rectangle(sourceX, 0, sphere.width, sphere.height),
                                 new Point(0, 0));

    event.updateAfterEvent();
}
```

El código realiza tres operaciones:

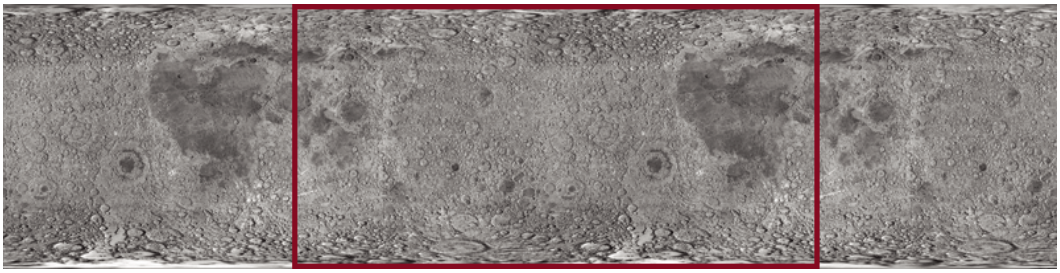
1 El valor de la variable `sourceX` (establecido inicialmente en 0) se incrementa en 1.

```
sourceX += 1;
```

Tal y como se puede observar, `sourceX` se usa para determinar la ubicación en `textureMap` desde donde los píxeles se copiarán en `sphere`, por lo que este código tiene el efecto de mover el rectángulo un píxel a la derecha en `textureMap`. Volviendo a la representación visual, tras varios ciclos de animación, el rectángulo de origen se habrá movido varios píxeles hacia la derecha, tal y como se muestra a continuación:

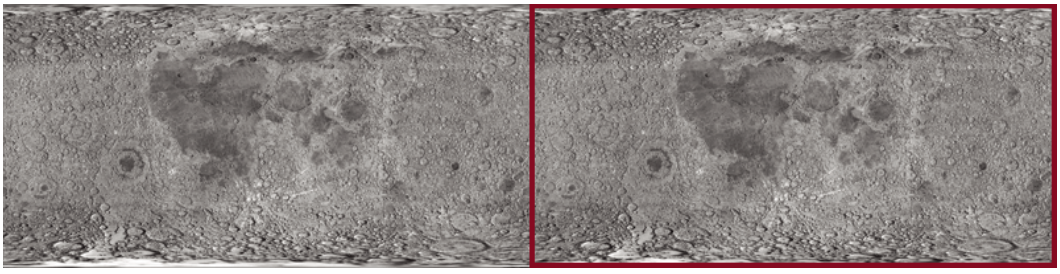


Tras varios ciclos más, el rectángulo se habrá movido incluso más lejos:



Este cambio constante y gradual en la ubicación desde la que se copian los píxeles es la clave en la animación. Con el movimiento lento y continuo de la ubicación de origen hacia la derecha, la imagen que se muestra en la pantalla en `sphere` parecer deslizarse continuamente hacia la izquierda. Por este motivo, es necesario que la imagen de origen (`textureMap`) tenga dos copias de la fotografía de la superficie de la luna. Debido a que el rectángulo se mueve continuamente hacia derecha, la mayor parte del tiempo no lo hace sobre una sola fotografía de la luna, sino que se superponen las dos fotografías.

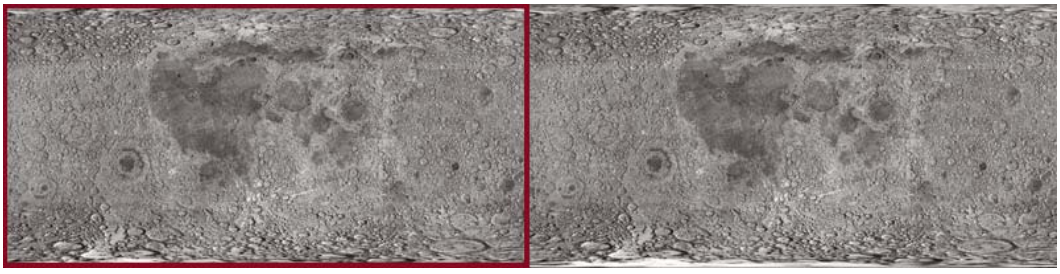
2 Con el movimiento lento del rectángulo de origen hacia la derecha, existe un problema. Finalmente el rectángulo llegará al borde derecho de `textureMap` y no dispondrá de píxeles de la fotografía de la luna para copiar en `sphere`:



En las siguientes líneas de código se aborda este problema:

```
if (sourceX >= textureMap.width / 2)
{
    sourceX = 0;
}
```

El código comprueba si `sourceX` (borde izquierdo del rectángulo) ha alcanzado la mitad de `textureMap`. Si es así, vuelve a restablecer `sourceX` a 0, moviéndolo de nuevo hacia el borde izquierdo de `textureMap` y volviendo a iniciar el ciclo:



3 Con el valor adecuado y calculado de `sourceX`, el paso final en la creación de la animación consiste en copiar los nuevos píxeles del rectángulo de origen en `sphere`. El código que realiza esta operación es muy similar al que llenó en un principio `sphere` (descrito anteriormente); en este caso la única diferencia es que en la llamada al constructor `new Rectangle()`, el borde izquierdo del rectángulo se sitúa en `sourceX`:

```
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(sourceX, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

Recuerde que este código se llama repetidamente, cada 15 milisegundos. Debido a que la ubicación del rectángulo de origen se desplaza continuamente y los píxeles se copian en `sphere`, la apariencia en pantalla es que la imagen de la fotografía de la luna representada mediante `sphere` se desliza constantemente. Es decir, la luna parece girar de forma continua.

Creación de la apariencia esférica

Obviamente, la luna es una esfera y no un rectángulo. Por lo tanto, en el ejemplo es necesario tomar la fotografía de la superficie de la luna rectangular, conforme se anima continuamente, y convertirla en una esfera. Esto implica dos pasos independientes: una máscara se utiliza para ocultar todo el contenido excepto una región circular de la fotografía de la superficie lunar y un filtro de mapa de desplazamiento se emplea para distorsionar la apariencia de la fotografía para hacerla parecer tridimensional.

En primer lugar, se usa una máscara con forma de círculo para ocultar todo el contenido del objeto MoonSphere excepto la esfera creada por el filtro. El siguiente código crea la máscara como una instancia de Shape y la aplica como la máscara de la instancia de MoonSphere:

```
moonMask = new Shape();
moonMask.graphics.beginFill(0);
moonMask.graphics.drawCircle(0, 0, radius);
this.addChild(moonMask);
this.mask = moonMask;
```

Se debe tener en cuenta que dado que MoonSphere es un objeto de visualización (se basa en la clase Sprite), la máscara se puede aplicar directamente a la instancia de MoonSphere utilizando su propiedad heredada `mask`.

Ocultar simplemente las partes de la fotografía utilizando una máscara con forma de círculo no es suficiente para crear un efecto de esfera giratoria con un aspecto realista. Debido al modo en que se tomó la fotografía de la superficie lunar, sus dimensiones no son proporcionales; las partes de la imagen que se encuentran más hacia la parte superior o inferior de la imagen están más distorsionadas y expandidas en comparación con las partes del ecuador. Para distorsionar la apariencia de la fotografía para hacerla parecer tridimensional, utilizaremos un filtro de mapa de desplazamiento.

Un filtro de mapa de desplazamiento es un tipo de filtro que se usa para distorsionar una imagen. En este caso, la fotografía de la luna se "distorsionará" para hacerla más realista, contrayendo la parte superior e inferior de la imagen horizontalmente, mientras el centro se deja sin cambio. Suponiendo que el filtro funcione en una parte con forma de cuadrado de la fotografía, la contracción de la parte superior e inferior pero no del centro convertirá el cuadrado en un círculo. Un efecto indirecto de la animación de esta imagen distorsionada es que el centro de la imagen parece moverse más lejos en la distancia de píxeles real que las áreas cercanas a la parte superior e inferior, lo que crea la ilusión de que el círculo es realmente un objeto tridimensional (una esfera).

El siguiente código se utiliza para crear el filtro de mapa de desplazamiento denominado `displaceFilter`:

```
var displaceFilter:DisplacementMapFilter;
displaceFilter = new DisplacementMapFilter(fisheyeLens,
                                new Point(radius, 0),
                                BitmapDataChannel.RED,
                                BitmapDataChannel.GREEN,
                                radius, 0);
```

El primer parámetro, `fisheyeLens`, se conoce como la imagen del mapa; en este caso se trata de un objeto BitmapData que se crea mediante programación. La creación de esta imagen se describe más adelante en la sección "Creación de una imagen de mapa de bits estableciendo valores de píxel" en la página 514. Los demás parámetros describen la posición en la imagen filtrada en la que se debe aplicar el filtro, qué canales de color se utilizarán para controlar el efecto de desplazamiento y hasta qué punto afectarán al desplazamiento. Una vez creado el filtro de mapa de desplazamiento, se aplica a `sphere`, aún en el método `imageLoadComplete()`:

```
sphere.filters = [displaceFilter];
```

La imagen final, con máscara y filtro de mapa de desplazamiento aplicados, presenta el siguiente aspecto:

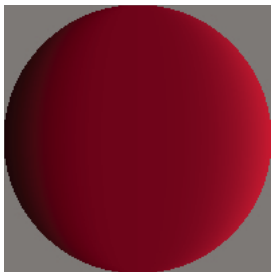
Con cada ciclo de la animación de rotación de la luna, el contenido de BitmapData de la esfera se sobrescribe con una nueva instantánea de los datos de la imagen de origen. No obstante, el filtro no necesita volver a aplicarse cada vez. Esto se debe a que el filtro se aplica en la instancia de Bitmap (el objeto de visualización) en lugar de en los datos del mapa de bits (información del píxel sin procesar). Recuerde que la instancia de Bitmap no son los datos de mapa de bits reales; se trata de un objeto de visualización que muestra los datos de mapa de bits en pantalla. Para utilizar una analogía, una instancia de Bitmap es como el proyector de diapositivas que se utiliza para visualizar diapositivas fotográficas en una pantalla y un objeto BitmapData es como la diapositiva real que se puede presentar mediante un proyector. Es posible aplicar un filtro directamente a un objeto BitmapData, lo que sería comparable a dibujar directamente en una diapositiva fotográfica para alterar la imagen. También se puede aplicar un filtro a cualquier objeto de visualización, incluyendo una instancia de Bitmap; esto sería como colocar un filtro frente a la lente del proyector de diapositivas para distorsionar el resultado que se muestra en pantalla (sin que la diapositiva original se vea alterada). Debido a que se puede acceder a los datos de mapa de bits mediante la propiedad bitmapData de una instancia de Bitmap, el filtro podría haberse aplicado directamente en los datos de mapa bits sin procesar. Sin embargo, en este caso resulta lógico aplicar el filtro al objeto de visualización Bitmap en lugar de a los datos de mapa de bits.

Para obtener información detallada sobre el uso del filtro de mapa de desplazamiento en ActionScript, consulte "Aplicación de filtros a objetos de visualización" en la página 363.

Creación de una imagen de mapa de bits estableciendo valores de píxel

Un aspecto importante del filtro de mapa de desplazamiento es que implica realmente dos imágenes. Una imagen, la imagen de origen, es la que se vé modificada por el filtro. En este ejemplo, la imagen de origen en la instancia de Bitmap denominada `sphere`. La otra imagen utilizada por el filtro se denomina imagen del mapa. La imagen del mapa no se visualiza realmente en pantalla. En cambio, el color de cada uno de sus píxeles se utiliza como una entrada en la función de desplazamiento; el color del píxel en una determinada coordenada x, y en la imagen del mapa determina el grado de desplazamiento (cambio físico de posición) que se aplica al píxel en dicha coordenada x, y en la imagen de origen.

Por lo tanto, para utilizar el filtro de mapa de desplazamiento para crear un efecto de esfera, el ejemplo necesita la imagen del mapa adecuada; una imagen con fondo gris y un círculo relleno con un degradado de un solo color (rojo) que pase horizontalmente de un tono oscuro a claro, tal y como se muestra a continuación:



Debido a que únicamente se utiliza un filtro y una imagen del mapa en este ejemplo, la imagen del mapa sólo se crea una sola vez, en el método `imageLoadComplete()` (es decir, cuando la imagen externa termina de cargarse). La imagen del mapa, denominada `fisheyeLens`, se crea llamando al método `createFisheyeMap()` de la clase MoonSphere:

```
var fisheyeLens:BitmapData = createFisheyeMap(radius);
```

Dentro del método `createFisheyeMap()`, la imagen del mapa se dibuja un píxel cada vez utilizando el método `setPixel()` de la clase BitmapData. El código completo para el método `createFisheyeMap()` se incluye a continuación, seguido de un análisis paso a paso sobre su funcionamiento:

```
private function createFisheyeMap(radius:int):BitmapData
{
    var diameter:int = 2 * radius;

    var result:BitmapData = new BitmapData(diameter,
                                           diameter,
                                           false,
                                           0x808080);

    // Loop through the pixels in the image one by one
    for (var i:int = 0; i < diameter; i++)
    {
        for (var j:int = 0; j < diameter; j++)
        {
            // Calculate the x and y distances of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctX:Number = (i - radius) / radius;
            var pctY:Number = (j - radius) / radius;

            // Calculate the linear distance of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);

            // If the current pixel is inside the circle,
            // set its color.
            if (pctDistance < 1)
            {
                // Calculate the appropriate color depending on the
                // distance of this pixel from the center of the circle.
                var red:int;
                var green:int;
                var blue:int;
                var rgb:uint;
                red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
                green = 0;
                blue = 0;
                rgb = (red << 16 | green << 8 | blue);
                // Set the pixel to the calculated color.
                result.setPixel(i, j, rgb);
            }
        }
    }
    return result;
}
```

En primer lugar, cuando se llama al método éste recibe un parámetro, `radius`, que indica el radio de la imagen de forma circular que se va a crear. A continuación, el código crea el objeto BitmapData en el que se dibujará el círculo. Dicho objeto, llamado `result`, se transmite finalmente como el valor devuelto del método. Tal y como se muestra en el siguiente fragmento de código, la instancia de BitmapData `result` se crea con una anchura y una altura similares al diámetro del círculo, sin transparencia (`false` para el tercer parámetro) y rellena previamente con el color `0x808080` (gris medio):

```
var result:BitmapData = new BitmapData(diameter,
                                       diameter,
                                       false,
                                       0x808080);
```

A continuación, el código utiliza dos bucles para repetir cada píxel de la imagen. El bucle exterior recorre todas las columnas de la imagen de izquierda a derecha (utilizando la variable `i` para representar la posición horizontal del píxel que se está manipulando en ese momento), mientras que el bucle interior recorre todos los píxeles de la columna actual desde la parte superior a la inferior (con la variable `j` que representa la posición vertical del píxel actual). El código de los bucles (donde se omite el contenido del bucle interior) se muestra a continuación:

```
for (var i:int = 0; i < diameter; i++)
{
    for (var j:int = 0; j < diameter; j++)
    {
        ...
    }
}
```

A medida que los bucles recorren los píxeles uno por uno, en cada píxel se calcula un valor (el valor del color del píxel en la imagen del mapa). Este proceso consta de cuatro pasos:

1. El código calcula la distancia del píxel actual desde el centro del círculo a través del eje x (`i - radius`). Dicho valor se divide por el radio para hacerlo un porcentaje del radio en lugar de una distancia absoluta (`(i - radius) / radius`). Ese valor del porcentaje se almacena en una variable denominada`pctX` y el valor equivalente para el eje y se calcula y almacena en la variable `pctY`, tal y como se indica en este código:

   ```
   var pctX:Number = (i - radius) / radius;
   var pctY:Number = (j - radius) / radius;
   ```

2. Utilizado una fórmula trigonométrica estándar, el teorema de Pitágoras, la distancia lineal entre el centro del círculo y el punto actual se calcula a partir de `pctX` y `pctY`. Dicho valor se almacena en una variable denominada `pctDistance`, tal y como se muestra a continuación:

   ```
   var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);
   ```

3. A continuación, el código comprueba si el porcentaje de distancia es inferior a 1 (lo que significa el 100% del radio; es decir, si el píxel que se considera está en el radio del ciclo). Si el píxel se encuentra dentro del círculo, se la asigna un valor de color calculado (se omite aquí, pero se describe en el paso 4); de lo contrario, no sucede nada más con ese píxel, de forma que su color se deja como gris medio predeterminado:

   ```
   if (pctDistance < 1)
   {
       ...
   }
   ```

4. Para los píxeles que se encuentren dentro del círculo, se calcula un valor de color. El color final será una tonalidad de rojo que comprende desde negro (0% rojo) en el borde izquierdo del círculo hasta rojo claro (100%) en el borde derecho del círculo. El valor de color se calcula en un principio en tres partes (rojo, verde y azul), tal y como se indica a continuación:

```
red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
green = 0;
blue = 0;
```

Se debe tener en cuenta que únicamente la parte roja del color (variable `red`) dispone realmente de un valor. Los valores de azul y verde (variables `green` y `blue`) se muestran aquí para mayor claridad, pero se pueden omitir. El objetivo de este método es crear un círculo que incluya un degradado de rojo, por lo que los valores de verde o azul no son necesarios.

Una vez que se determinen los tres valores de color independientes, se combinan en un solo valor de color entero utilizando un algoritmo estándar de desplazamiento de bits, tal y como se muestra en este código:

```
rgb = (red << 16 | green << 8 | blue);
```

Finalmente, con el valor de color calculado, dicho valor se asigna al píxel actual utilizando el método `setPixel()` del objeto BitmapData de `result`, tal y como se indica a continuación:

```
result.setPixel(i, j, rgb);
```

Capítulo 23: Trabajo en tres dimensiones (3D)

Fundamentos del concepto 3D

Introducción al concepto 3D en ActionScript

La principal diferencia entre un objeto bidimensional (2D) y otro tridimensional (3D) proyectado en una pantalla bidimensional es la adición de una tercera dimensión en el objeto. La tercera dimensión permite que el objeto se acerque y se aleje del punto de vista del usuario.

Si la propiedad `z` de un objeto de visualización se establece de forma explícita en un valor numérico, el objeto crea automáticamente una matriz de transformación 3D. Esta matriz se puede alterar para modificar la configuración de la transformación 3D del objeto.

Asimismo, la rotación 3D difiere de la rotación bidimensional. En 2D, el eje de rotación siempre es perpendicular al plano x/y; es decir, en el eje z. En 3D, el eje de rotación puede estar alrededor de cualquiera de los ejes x, y o z. La definición de las propiedades de escala y rotación de un objeto de visualización permite moverlo en un espacio 3D.

Tareas 3D comunes

En este capítulo se describen las siguientes tareas comunes relacionadas con el concepto 3D:

- Creación de un objeto 3D
- Movimiento de un objeto en el espacio 3D
- Rotación de un objeto en el espacio 3D
- Representación de la profundidad con el uso de la proyección en perspectiva
- Reordenamiento de la lista de visualización para que se corresponda con los ejes z relativos, de forma que los objetos se muestren correctamente unos delante de otros para el espectador.
- Transformación de objetos 3D con el uso de matrices 3D
- Utilización de vectores para manipular objetos en el espacio 3D
- Uso del método `Graphics.drawTriangles()` para crear perspectiva
- Utilización del mapeado UV para añadir texturas de mapas de bits a un objeto 3D
- Establecimiento del parámetro "culling" para la eliminación de superficies ocultas del método `Graphics.drawTriangles()` para agilizar la representación y ocultar partes de un objeto 3D que no se encuentran visibles desde el punto de vista actual.

Términos y conceptos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Perspectiva: en un plano 2D, representación de líneas paralelas convergentes en un punto de fuga para crear la ilusión de profundidad y distancia.

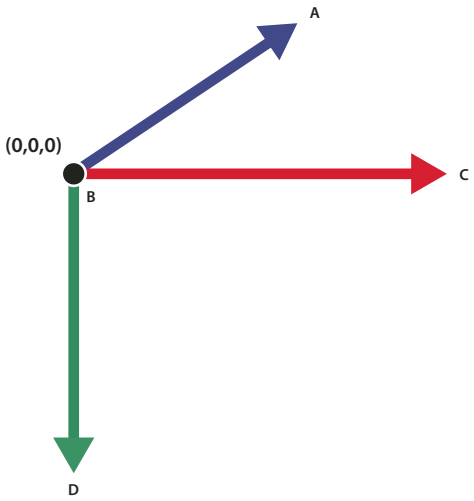
- Proyección: producción de una imagen 2D de un objeto de mayores dimensiones; los mapas en proyección 3D señalan a un plano 2D.
- Rotación: cambio de la orientación (y a menudo de la posición) de un objeto moviendo todos los puntos incluidos en el objeto en un movimiento circular.
- Transformación: alteración de puntos 3D o de conjuntos de puntos mediante la traslación, escala, rotación y sesgo o una combinación de estas operaciones.
- Traslación: cambio de posición de un objeto moviendo todos los puntos incluidos en el objeto mediante la misma cantidad en la misma dirección.
- Punto de fuga: punto en el que las líneas paralelas que se alejan parecen coincidir cuando se representan en una perspectiva lineal.
- Vector: un vector 3D representa un punto o una ubicación en el espacio tridimensional mediante el uso de las coordenadas cartesianas x, y y z.
- Vértice: un punto de esquina.
- Malla con textura: cualquier punto que defina un objeto en el espacio 3D.
- Mapeado UV: una forma de aplicar una textura o mapa de bits a una superficie 3D. El mapeado UV asigna valores a las coordenadas de una imagen como porcentajes del eje (U) horizontal y el eje (V) vertical.
- Valor T: factor de escala para determinar el tamaño de un objeto 3D conforme el objeto se acerca o se aleja del punto de vista actual.
- Culling: representación, o no, de superficies, con una dirección de trazado específica. Con el uso de la técnica "culling" se pueden ocultar superficies que no se encuentran visibles en el punto de vista actual.

Aspectos básicos de las funciones 3D de Flash Player y el tiempo de ejecución de AIR

En versiones de Flash Player anteriores a Flash Player 10 y en versiones de Adobe AIR anteriores a Adobe AIR 1.5, los objetos de visualización tienen dos propiedades, `x` e `y`, que permiten colocarlos en un plano bidimensional. A partir de Flash Player 10 y Adobe AIR 1.5, todos los objetos de visualización de ActionScript tienen una propiedad `z` que permite colocarlos a lo largo del eje z (eje que suele usarse para indicar profundidad o distancia).

Flash Player 10 y Adobe AIR 1.5 incorporan compatibilidad para efectos 3D. No obstante, los objetos de visualización son esencialmente planos. Finalmente todos los objetos de visualización como, por ejemplo, MovieClip o Sprite, se representan a sí mismos en dos dimensiones, en un solo plano. Las funciones de 3D permiten situar, mover, rotar y transformar estos objetos planos en tres dimensiones. También permiten administrar puntos 3D y convertirlos en coordenadas x, y 2D para que pueda proyectar objetos 3D en una vista bidimensional. Con el uso de estas funciones puede simular diversos tipos de experiencias en 3D.

El sistema de coordenadas 3D utilizado por ActionScript difiere de otros sistemas. Cuando se utilizan coordenadas 2D en ActionScript, el valor de x aumenta conforme se desplaza hacia la derecha por el eje x y el valor de y aumenta conforme desciende por el eje y. El sistema de coordenadas 3D conserva estas convenciones y añade un eje z cuyo valor aumenta conforme se aleja del punto de vista.



Direcciones positivas de los ejes x, y y z en el sistema de coordenadas 3D de ActionScript.
***A.** Eje + Z **B.** Origen **C.** +Eje X **D.** +Eje Y*

***Nota:** se debe tener en cuenta que Flash Player y AIR siempre representan 3D en capas. Esto significa que si el objeto A está delante del objeto B en la lista de visualización, Flash Player o AIR siempre representan A delante de B independientemente de los valores del eje z de los dos objetos. Para resolver este conflicto entre el orden de la lista de visualización y el orden del eje z, utilice el método* `transform.getRelativeMatrix3D()` *para guardar y posteriormente reordenar las capas de los objetos de visualización 3D. Para obtener más información, consulte* "Utilización de objetos Matrix3D para reordenar la visualización" en la página 529.

Las siguientes clases de ActionScript admiten las nuevas funciones relacionadas con 3D:

1. La clase flash.display.DisplayObject contiene la propiedad `z` y nuevas propiedades de escala y rotación para manipular objetos de visualización en el espacio 3D. El método `DisplayObject.local3DToGlobal()` ofrece una forma sencilla de proyectar geometría 3D en un plano 2D.
2. La clase flash.geom.Vector3D se puede utilizar como estructura de datos para administrar puntos 3D. También admite matemáticas vectoriales.
3. La clase flash.geom.Matrix3D admite transformaciones complejas de geometría 3D como, por ejemplo, rotación, escala y traslación.
4. La clase flash.geom.PerspectiveProjection controla los parámetros para el mapeado de geometría 3D en una vista 2D.

Existen dos enfoques diferentes en la simulación de imágenes 3D en ActionScript:

1. Organización y animación de objetos planos en un espacio 3D. Este enfoque implica la animación de objetos de visualización utilizando las propiedades x, y y z de los objetos, o bien, estableciendo propiedades de rotación y escala mediante la clase DisplayObject. Se puede obtener un movimiento más complejo utilizando el objeto DisplayObject.transform.matrix3D. El objeto DisplayObject.transform.perspectiveProjection personaliza el modo en que los objetos de visualización se dibujan en perspectiva 3D. Utilice este enfoque cuando desee animar objetos 3D que consten de planos fundamentalmente. Los ejemplos de este enfoque incluyen galerías de imágenes 3D u objetos de animación 2D organizados en un espacio 3D.

2 Generación de triángulos 2D a partir de geometría 3D y representación de estos triángulos con texturas. Para utilizar este enfoque, en primer lugar debe definir y administrar datos sobre objetos 3D y posteriormente convertir estos datos en triángulos 2D para la representación. Las texturas de mapa de bits se pueden asignar a estos triángulos y a continuación los triángulos se dibujan en un objeto graphics utilizando el método `Graphics.drawTriangles()`. Los ejemplos de este enfoque incluyen la carga de datos del modelo 3D desde un archivo y la representación del modelo en pantalla, o bien, la generación y el dibujo de terreno 3D como mallas de triángulo.

Creación y movimiento de objetos 3D

Para convertir un objeto de visualización 2D en un objeto de visualización 3D, puede establecer su propiedad `z` de forma explícita en un valor numérico. Cuando se asigna un valor a la propiedad `z`, se crea un nuevo objeto Transform para el objeto de visualización. Con la definición de las propiedades `DisplayObject.rotationX` o `DisplayObject.rotationY` también se crea un nuevo objeto Transform. El objeto Transform contiene una propiedad `Matrix3D` que determina el modo en que se representa el objeto de visualización en el espacio 3D.

En el siguiente código se establecen las coordenadas para un objeto de visualización denominado "leaf":

```
leaf.x = 100; leaf.y = 50; leaf.z = -30;
```

Estos valores, así como las propiedades que se derivan de los mismos, se pueden ver en la propiedad `matrix3D` del objeto Transform de la hoja:

```
var leafMatrix:Matrix3D  = leaf.transform.matrix3D;

trace(leafMatrix.position.x);
trace(leafMatrix.position.y);
trace(leafMatrix.position.z);
trace(leafMatrix.position.length);
trace(leafMatrix.position.lengthSquared);
```

Para obtener información sobre las propiedades del objeto Transform, consulte la clase Transform. Para obtener información sobre las propiedades del objeto Matrix3D, consulte la clase Matrix3D.

Movimiento de un objeto en el espacio 3D

Puede mover un objeto en un espacio 3D cambiando los valores de sus propiedades `x`, `y` o `z`. Cuando se cambia el valor de la propiedad `z`, el objeto parece acercarse o alejarse del espectador.

El siguiente código mueve dos elipses hacia delante y hacia atrás a lo largo de sus ejes z, cambiando el valor de sus propiedades `z` como respuesta a un evento. `ellipse2` se mueve más rápido que `ellipse1`: su propiedad `z` se vé aumentada por un múltiplo de 20 en cada evento Frame, mientras que la propiedad `z` de `ellipse1` se vé aumentada por un múltiplo de 10:

```
var depth:int = 1000;

function ellipse1FrameHandler(e:Event):void
{
    ellipse1Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 10;
}
function ellipse2FrameHandler(e:Event):void
{
    ellipse2Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 20;
}
function setDepth(e:Event, d:int):int
{
    if(e.currentTarget.z > depth)
    {
        e.currentTarget.z = depth;
        d = -1;
    }
    else if (e.currentTarget.z <  0)
    {
        e.currentTarget.z = 0;
        d = 1;
    }
}
```

Rotación de un objeto en el espacio 3D

Se puede girar un objeto de tres formas distintas, dependiendo del modo en que se establezcan las propiedades de rotación del objeto: `rotationX`, `rotationY` y `rotationZ`.

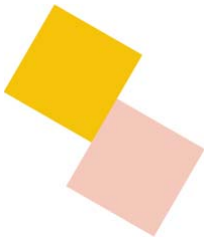
La siguiente figure muestra dos cuadrados que no se giran:

La siguiente figura muestra los dos cuadrados cuando se incrementa la propiedad `rotationY` del contenedor de los cuadrados para girarlos en el eje y. Con la rotación del contenedor, u objeto de visualización principal, de los dos cuadrados giran ambos cuadrados:

```
container.rotationY += 10;
```

En la siguiente figura se muestra qué sucede cuando se establece la propiedad `rotationX` del contenedor de los cuadrados. Con ello se giran los cuadrados en el eje x.

En la siguiente figura se muestra qué sucede cuando se incrementa la propiedad `rotationZ` del contenedor de los cuadrados. Con ello se giran los cuadrados en el eje z.



Un objeto de visualización se puede mover y girar simultáneamente en un espacio 3D.

Proyección de objetos 3D en una vista bidimensional

La clase PerspectiveProjection del paquete `flash.geom` proporciona una forma sencilla de aplicar una perspectiva básica al mover objetos de visualización en un espacio 3D.

Si no crea de forma explícita una proyección en perspectiva para el espacio 3D, el motor 3D utiliza un objeto PerspectiveProjection predeterminado que se encuentra en la raíz y se propaga a todos sus elementos secundarios.

Las tres propiedades que definen cómo muestra un objeto PerspectiveProjection el espacio 3D son las siguientes:

- `fieldOfView`
- `projectionCenter`
- `focalLength`

Con la modificación del valor de `fieldOfView`, cambia automáticamente el valor de `focalLength` y viceversa, ya que son interdependientes.

La fórmula utilizada para calcular `focalLength` dado el valor de `fieldOfView` es la siguiente:

```
focalLength = stageWidth/2 * (cos(fieldOfView/2) / sin(fieldOfView/2)
```

Generalmente se modificaría la propiedad `fieldOfView` de forma explícita.

Campo visual

Con la manipulación de la propiedad `fieldOfView` de la clase PerspectiveProjection, puede hacer que un objeto de visualización 3D que se acerca al espectador parezca más grande y que un objeto que se aleja del espectador parezca más pequeño.

La propiedad `fieldOfView` especifica un ángulo entre 0 y 180 grados que determina la intensidad de la proyección en perspectiva. Cuanto mayor es el valor, mayor es la distorsión aplicada a un objeto de visualización que se mueve por su eje z. Un valor `fieldOfView` bajo implica un ajuste de escala muy pequeño y hace que objeto sólo parezca moverse ligeramente hacia atrás en el espacio. Un valor `fieldOfView` alto causa más distorsión y la apariencia de mayor movimiento. El valor máximo de 180 grados tiene como resultado un efecto de lente de cámara "ojo de pez" extremo.

Centro de proyección

La propiedad `projectionCenter` representa el punto de fuga en la proyección en perspectiva. Se aplica como desplazamiento en el punto de registro predeterminado (0,0) en la esquina superior izquierda del escenario.

Cuando un objeto parece alejarse del espectador, se sesga hacia el punto de fuga y finalmente desaparece. Imagine un pasillo infinitamente largo. Cuando mira el pasillo hacia delante, los bordes de las paredes convergen en un punto de fuga hacia el final del pasillo.

Si el punto de fuga está en el centro del escenario, el pasillo desaparece hacia un punto en el centro. El valor predeterminado de la propiedad `projectionCenter` es el centro del escenario. Si, por ejemplo, desea que los elementos aparezcan en la parte izquierda del escenario y un área 3D a la derecha, establezca `projectionCenter` en un punto en la derecha del escenario para que ese sea el punto de fuga del área de visualización 3D.

Distancia focal

La propiedad `focalLength` representa la distancia entre el origen del punto de vista (0,0,0) y la ubicación del objeto de visualización en su eje z.

Una distancia focal larga es como una lente telefoto con una vista estrecha y distancias comprimidas entre objetos. Una distancia focal corta es como una lente gran angular, con la que se obtiene una vista amplia con mucha distorsión. La distancia focal media se aproxima a lo que vé el ojo humano.

Generalmente el valor `focalLength` se vuelve a calcular de forma dinámica durante la transformación de perspectiva conforme se mueve el objeto de visualización, aunque se puede establecer de forma explícita.

Valores de proyección en perspectiva predeterminados

El objeto predeterminado PerspectiveProjection creado en la raíz tiene los siguientes valores:

- `fieldOfView: 55`
- `perspectiveCenter: stagewidth/2, stageHeight/2`
- `focalLength: stageWidth/ 2 * ( cos(fieldOfView/2) / sin(fieldOfView/2) )`

Estos son los valores que se utilizan si no crea su propio objeto PerspectiveProjection.

Puede crear una instancia de su propio objeto PerspectiveProjection con el fin realizar una modificación propia de `projectionCenter` y `fieldOfView`. En este caso, los valores predeterminados del objeto recién creado son los siguientes, en función del tamaño del escenario predeterminado de 500 por 500:

- `fieldOfView: 55`
- `perspectiveCenter: 250,250`
- `focalLength: 480.24554443359375`

Ejemplo: Proyección en perspectiva

En el siguiente ejemplo se muestra el uso de la proyección en perspectiva para crear espacio 3D. Muestra cómo se puede modificar el punto de fuga y cambiar la proyección en perspectiva del espacio mediante la propiedad `projectionCenter`. Esta modificación implica volver a calcular los valores `focalLength` y `fieldOfView` con su distorsión añadida del espacio 3D.

En este ejemplo:

1. Se crea el objeto objeto denominado `center`, como un círculo con cruz.
2. Asigna la coordenadas del objeto Sprite `center` a la propiedad`projectionCenter` de la propiedad `perspectiveProjection` de `transform` de la raíz.
3. Se añaden detectores de eventos para eventos de ratón que llaman a controladores que modifican el componente `projectionCenter` de forma que éste siga la ubicación del objeto `center`.
4. Se crean cuatro cuadros de estilo Accordion que forman las paredes del espacio en perspectiva.

Cuando pruebe este ejemplo, ProjectionDragger.swf, arrastre el círculo alrededor de las distintas ubicaciones. El punto de fuga sigue el círculo, situándose donde lo coloque. Observe los cuadros que delimitan la extensión del espacio y aplique la distorsión cuando aleje el centro de proyección del centro del escenario.

Para obtener los archivos de la aplicación para esta muestra, vaya a www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación ProjectionDragger se encuentran en la carpeta Samples/ProjectionDragger.

```
package
{
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.geom.Point;
    import flash.events.*;
    public class ProjectionDragger extends Sprite
    {
        private var center : Sprite;
        private var boxPanel:Shape;
        private var inDrag:Boolean = false;

        public function ProjectionDragger():void
        {
            createBoxes();
            createCenter();
        }
        public function createCenter():void
        {
            var  centerRadius:int = 20;

            center = new Sprite();

            // circle
            center.graphics.lineStyle(1, 0x000099);
            center.graphics.beginFill(0xCCCCCC, 0.5);
            center.graphics.drawCircle(0, 0, centerRadius);
            center.graphics.endFill();
            // cross hairs
            center.graphics.moveTo(0, centerRadius);
            center.graphics.lineTo(0, -centerRadius);
```

```
            center.graphics.moveTo(centerRadius, 0);
            center.graphics.lineTo(-centerRadius, 0);
            center.x = 175;
            center.y = 175;
            center.z = 0;
            this.addChild(center);

            center.addEventListener(MouseEvent.MOUSE_DOWN, startDragProjectionCenter);
            center.addEventListener(MouseEvent.MOUSE_UP, stopDragProjectionCenter);
            center.addEventListener( MouseEvent.MOUSE_MOVE, doDragProjectionCenter);
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
        }
        public function createBoxes():void
        {
            // createBoxPanel();
            var boxWidth:int = 50;
            var boxHeight:int = 50;
            var numLayers:int = 12;
            var depthPerLayer:int = 50;

            // var boxVec:Vector.<Shape> = new Vector.<Shape>(numLayers);
            for (var i:int = 0; i < numLayers; i++)
            {
                this.addChild(createBox(150, 50, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCCCFF));
                this.addChild(createBox(50, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xFFCCCC));
                this.addChild(createBox(250, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCFFCC));
                this.addChild(createBox(150, 250, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xDDDDDD));
            }
        }

        public function createBox(xPos:int = 0, yPos:int = 0, zPos:int = 100, w:int = 50, h:int
= 50, color:int = 0xDDDDDD):Shape
        {
            var box:Shape = new Shape();
            box.graphics.lineStyle(2, 0x666666);
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, w, h);
            box.graphics.endFill();
            box.x = xPos;
            box.y = yPos;
            box.z = zPos;
            return box;
        }
        public function startDragProjectionCenter(e:Event)
        {
            center.startDrag();
            inDrag = true;
```

```
        }

        public function doDragProjectionCenter(e:Event)
        {
            if (inDrag)
            {
                root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            }
        }

        public function stopDragProjectionCenter(e:Event)
        {
            center.stopDrag();
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            inDrag = false;
        }
    }
}
```

Para obtener proyecciones en perspectiva más complejas, utilice la clase Matrix3D.

Transformaciones 3D complejas

La clase Matrix3D permite transformar puntos 3D en un espacio de coordenadas o mapear puntos 3D de un espacio de coordenadas a otro.

Para utilizar la clase Matrix3D no es necesario conocer las matemáticas de matrices. La mayoría de las operaciones comunes de transformación se pueden controlar utilizando los métodos de la clase. No tiene que preocuparse por establecer o calcular de forma explícita los valores de todos los elementos de la matriz.

Una vez establecida la propiedad `z` de un objeto de visualización en un valor numérico, puede recuperar la matriz de transformación del objeto utilizando la propiedad Matrix3D del objeto Transform del objeto de visualización:

```
var leafMatrix:Matrix3D = this.transform.matrix3D;
```

Puede utilizar los métodos del objeto Matrix3D para realizar la traslación, rotación, escala y proyección en perspectiva del objeto de visualización.

Utilice la clase Vector3D con sus propiedades `x`, `y`, y `z` para administrar puntos 3D. También puede representar un vector espacial en física, que cuenta con una dirección y una magnitud. Los métodos de la clase Vector3D permiten realizar cálculos comunes con vectores espaciales como, por ejemplo, cálculos de producto escalar, producto cruzado y suma.

Nota: *la clase Vector3D no se relaciona con la clase Vector de ActionScript. La clase Vector3D contiene propiedades y métodos para definir y manipular puntos 3D, mientras que la clase Vector admite conjuntos de objetos con tipo.*

Creación de objetos Matrix3D

Existen tres formas principales de crear o recuperar objetos `Matrix3D`:

1 Utilice el método constructor `Matrix3D()` para crear una instancia de una nueva matriz. El constructor `Matrix3D()` adopta un objeto `Vector` que contiene 16 valores numéricos y sitúa cada valor en una celda de la matriz. Por ejemplo:

```
var rotateMatrix:Matrix3D = new Matrix3D(1,0,0,1, 0,1,0,1, 0,0,1,1, 0,0,0,1);
```

2 Establezca el valor de la propiedad `z` de un objeto de visualización. A continuación recupere la matriz de transformación de la propiedad `transform.matrix3D` de ese objeto.

3 Recupere el objeto Matrix3D que controla la visualización de objetos 3D en el escenario, obteniendo el valor de la propiedad `perspectiveProjection.matrix3D` del objeto de visualización raíz.

Aplicación de varias transformaciones 3D

Puede aplicar varias transformaciones 3D a la vez utilizando un objeto Matrix3D. Por ejemplo, si desea girar, escalar y mover un cubo, puede aplicar tres transformaciones independientes a cada punto del cubo. No obstante, resulta mucho más eficaz calcular previamente varias transformaciones en un objeto Matrix3D y posteriormente realizar una transformación de matriz en cada uno de los puntos.

Nota: *el orden en que se aplican las transformaciones de matriz es importante. Los cálculos de matriz no son conmutativos. Por ejemplo, con la aplicación de una rotación seguida de una traslación se obtiene un resultado diferente de la aplicación de la misma traslación seguida de la misma rotación.*

En el siguiente ejemplo se muestran dos formas de realizar varias transformaciones 3D.

```
package {
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;
    import flash.geom.*;

public class Matrix3DTransformsExample extends Sprite
    {
        private var rect1:Shape;
        private var rect2:Shape;

public function Matrix3DTransformsExample():void
        {
            var pp:PerspectiveProjection = this.transform.perspectiveProjection;
            pp.projectionCenter = new Point(275,200);
            this.transform.perspectiveProjection = pp;

            rect1 = new Shape();
            rect1.x = -70;
            rect1.y = -40;
            rect1.z = 0;
            rect1.graphics.beginFill(0xFF8800);
            rect1.graphics.drawRect(0,0,50,80);
            rect1.graphics.endFill();
            addChild(rect1);

            rect2 = new Shape();
            rect2.x = 20;
            rect2.y = -40;
            rect2.z = 0;
            rect2.graphics.beginFill(0xFF0088);
            rect2.graphics.drawRect(0,0,50,80);
            rect2.graphics.endFill();
            addChild(rect2);
```

```
            doTransforms();
        }

        private function doTransforms():void
        {
            rect1.rotationX = 15;
            rect1.scaleX = 1.2;
            rect1.x += 100;
            rect1.y += 50;
            rect1.rotationZ = 10;

            var matrix:Matrix3D = rect2.transform.matrix3D;
            matrix.appendRotation(15, Vector3D.X_AXIS);
            matrix.appendScale(1.2, 1, 1);
            matrix.appendTranslation(100, 50, 0);
            matrix.appendRotation(10, Vector3D.Z_AXIS);
            rect2.transform.matrix3D = matrix;
        }
    }
}
```

En el método `doTransforms()` el primer bloque de código utiliza las propiedades DisplayObject para cambiar la rotación, escala y posición de una forma de rectángulo. El segundo bloque de código utiliza los métodos de la clase Matrix3D para realizar las mismas transformaciones.

La principal ventaja del uso del método `Matrix3D` es que todos los cálculos se realizan en la matriz en primer lugar. Después se aplican al objeto de visualización sólo una vez, cuando se establece su propiedad `transform.matrix3D`. La definición de las propiedades DisplayObject hace que el código fuente resulte un poco más sencillo de leer. No obstante, cada vez que se establece una propiedad de escala o rotación, esto causa varios cálculos y cambia varias propiedades del objeto de visualización.

Si el código aplica las mismas transformaciones completas a los objetos de visualización varias veces, guarde el objeto Matrix3D como una variable y posteriormente vuelva a aplicarlo una y otra vez.

Utilización de objetos Matrix3D para reordenar la visualización

Tal y como se indicó anteriormente, el orden de la disposición en capas de los objetos en la lista de visualización determina el orden de las capas de visualización, independientemente de sus ejes z relativos. Si la animación transforma las propiedades de los objetos de visualización en un orden que difiere del de la lista de visualización, el espectador podría ver una disposición en capas de los objetos que no se corresponde con las capas del eje z. Por lo tanto, un objeto que deba aparecer alejado del espectador puede aparecer frente a un objeto que esté más cerca del espectador.

Para garantizar que la disposición en capas de los objetos de visualización 3D se corresponde con las profundidades relativas de los objetos, utilice un enfoque como el siguiente:

1. Utilice el método `getRelativeMatrix3D()` del objeto Transform para obtener los `ejes z` relativos de los objetos de visualización 3D secundarios.
2. Use el método `removeChild()` para eliminar los objetos de la lista de visualización.
3. Ordene los objetos de visualización en función de sus valores de eje z.
4. Utilice el método `addChild()` para volver a añadir los elementos secundarios a la lista de visualización en orden inverso.

Este reordenamiento garantiza que los objetos se muestren conforme a sus ejes z relativos.

El siguiente código muestra la correcta visualización de las seis caras de un cuadro 3D. Reordena las caras del cuadro tras haber aplicado las rotaciones:

```
public var faces:Array; . . .

public function ReorderChildren()
{
    for(var ind:uint = 0; ind < 6; ind++)
    {
        faces[ind].z = faces[ind].child.transform.getRelativeMatrix3D(root).position.z;
        this.removeChild(faces[ind].child);
    }
    faces.sortOn("z", Array.NUMERIC | Array.DESCENDING);
    for (ind = 0; ind < 6; ind++)
    {
        this.addChild(faces[ind].child);
    }
}
```

Para obtener los archivos de la aplicación para esta muestra, vaya a www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación se encuentran en la carpeta Samples/ReorderByZ.

Utilización de triángulos para efectos 3D

En ActionScript, las transformaciones de mapa de bits se realizan utilizando el método `Graphics.drawTriangles()`, ya que los modelos 3D se representan mediante una colección de triángulos en el espacio. (No obstante, Flash Player y AIR no admiten un búfer de profundidad, por lo que los objetos de visualización aún son esencialmente planos o bidimensionales. Esto se describe en "Aspectos básicos de las funciones 3D de Flash Player y el tiempo de ejecución de AIR" en la página 519.) El método `Graphics.drawTriangles()` es similar a `Graphics.drawPath()`, ya que adopta un conjunto de coordenadas para dibujar un trazado de triángulo.

Para familiarizarse con el uso de `Graphics.drawPath()`, consulte "Trazados de dibujo" en la página 343.

El método `Graphics.drawTriangles()` utiliza Vector.<Number> para especificar las ubicaciones de punto para el trazado del triángulo:

```
drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null, uvtData:Vector.<Number>
= null, culling:String = "none"):void
```

El primer parámetro de `drawTriangles()` es el único necesario: el parámetro `vertices`. Este parámetro es un vector de números que define las coordenadas mediante las que se dibujan los triángulos. Cada tres conjuntos de coordenadas (seis números) se representa un trazado del triángulo. Sin el parámetro `indices`, la longitud del vector siempre debe ser un factor de seis, ya que cada triángulo requiere tres pares de coordenadas(tres conjuntos de dos valores x/y). Por ejemplo:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([
        10,10,  100,10,  10,100,
        110,10, 110,100, 20,100]));
```

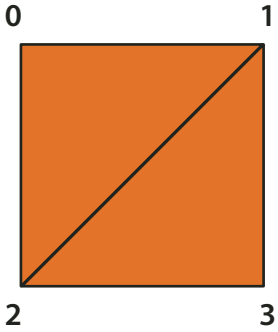
Ninguno de estos triángulos comparte puntos, pero si lo hicieran, el segundo parámetro `drawTriangles()`, `indices`, podría utilizarse para volver a usar los valores en el vector `vertices` para varios triángulos.

Al utilizar el parámetro `indices`, se debe tener en cuenta que los valores `indices` son índices de punto, y no índices que se relacionan directamente con los elementos del conjunto `vertices`. Es decir, un índice del vector `vertices` tal y como se define mediante `indices` es realmente el verdadero índice dividido por 2. Para el tercer punto de un vector `vertices` utilice, por ejemplo, un valor `indices` de 2, aunque el primer valor numérico de ese punto comience en el índice vectorial de 4.

Por ejemplo, combine dos triángulos para que compartan el borde diagonal utilizando el parámetro `indices`:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([10,10, 100,10, 10,100, 100,100]),
    Vector.<int>([0,1,2, 1,3,2]));
```

Se debe tener en cuenta que aunque ahora se haya dibujado un cuadrado utilizando dos triángulos, sólo se especificaron cuatro puntos en el vector `vertices`. Con el uso de `indices`, los dos puntos compartidos por los dos triángulos se volverán a utilizar para cada triángulo. Con ello se reduce el recuento total de vértices de 6 (12 números) a 4 (8 números):



Cuadrado dibujado con dos triángulos utilizando el parámetro de vértices

Esta técnica resulta útil con mallas de triángulo más grandes donde varios triángulos comparten la mayoría de los puntos.

Todos los rellenos se pueden aplicar a triángulos. Los rellenos se aplican a la malla de triángulo resultante tal y como se haría en cualquier otra forma.

Transformación de mapas de bits

Las transformaciones de mapa de bits proporcionan la ilusión de perspectiva o "textura" en un objeto tridimensional. Concretamente, es posible distorsionar un mapa de bits hacia un punto de fuga de forma que la imagen parezca encogerse conforme se acerca al punto de fuga. O bien, puede utilizar un mapa de bits bidimensional para crear una superficie para un objeto tridimensional, ofreciendo la ilusión de textura o “envoltura” en dicho objeto.

Superficie bidimensional donde se utiliza un punto de fuga y tres objetos tridimensionales ajustados con un mapa de bits

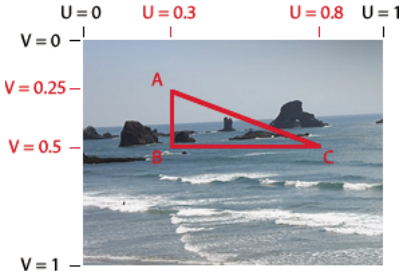
Mapeado UV

Una vez que comience a trabajar con texturas, deseará utilizar el parámetro uvtData de `drawTriangles()`. Este parámetro permite configurar el mapeado UV para rellenos de mapa de bits.

El mapeado UV es un método para aplicar textura a los objetos. Se basa en dos valores: un valor U horizontal (x) y un valor V vertical (y). En lugar de basarse en valores de píxel, se basan en porcentajes. 0 U y 0 V representan la parte superior izquierda de una imagen y 1 U y 1 V la parte inferior derecha:

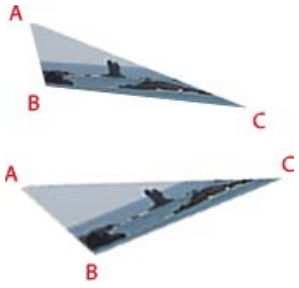
Ubicaciones UV 0 y 1 en una imagen de mapa de bits

A los vectores de un triángulo se les puede proporcionar coordenadas UV para asociarse con las ubicaciones respectivas en una imagen:



Coordenadas UV de un área triangular de una imagen de mapa de bits

Los valores UV coinciden con los puntos del triángulo:



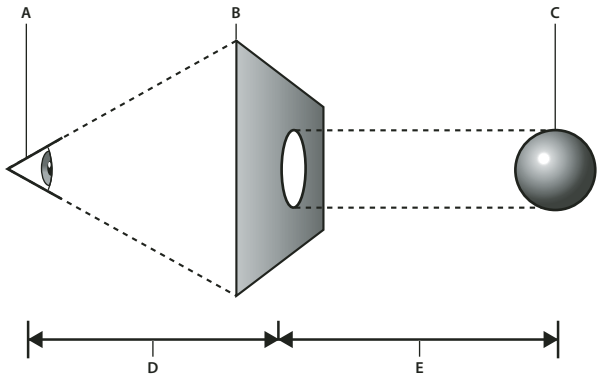
Los vértices del triángulo se mueven y el mapa de bits se distorsiona para mantener igual los valores UV para un punto individual

Debido a que las transformaciones 3D de ActionScript se aplican al triángulo asociado con el mapa de bits, la imagen del mapa de bits se aplica al triángulo en función de los valores UV. Por lo tanto, en lugar de utilizar cálculos de matriz, establezca o ajuste los valores UV para crear un efecto tridimensional.

El método `Graphics.drawTriangles()` también acepta una parte de información opcional para transformaciones tridimensionales: el valor T. El valor T en uvtData representa la perspectiva 3D, o más concretamente, el factor de escala del vértice asociado. El mapeado UVT añade una corrección de la perspectiva al mapeado UV. Por ejemplo, si un objeto se coloca en un espacio 3D alejado del punto de vista de forma que parezca ser un 50% de su tamaño "original", el valor T del objeto podría ser 0,5. Debido a que los triángulos se dibujan para representar objetos en el espacio 3D, sus ubicaciones en el eje z determinan sus valores T. La ecuación que determina el valor T es la siguiente:

```
T = focalLength/(focalLength + z);
```

En esta ecuación, focalLength representa una distancia focal o ubicación de "pantalla" calculada que determina la cantidad de perspectiva que se ofrece en la vista.



Distancia focal y valor z
***A.** punto de vista **B.** pantalla **C.** Objeto 3D **D.** valor focalLength **E.** valor z*

El valor de T se utiliza para realizar un ajuste de escala de formas básicas para hacerlas parecer más alejadas en la distancia. Suele ser el valor utilizado para convertir puntos 3D en puntos 2D. En el caso de datos UVT, también se utiliza para escalar un mapa de bits entre los puntos en un triángulo con perspectiva.

Cuando se definen valores UVT, el valor T sigue directamente los valores UV definidos para un vértice. Con la inclusión de T, cada tres valores en el parámetro `uvtData` (U, V y T) coinciden con cada dos valores del parámetro `vertices` (x e y). Sólo con los valores UV, uvtData.length == vertices.length. Con la inclusión de un valor T, uvtData.length = 1.5*vertices.length.

En el siguiente ejemplo se muestra un plano giratorio en un espacio 3D utilizando datos UVT. Este ejemplo utiliza una imagen llamada ocean.jpg y una clase "auxiliar", ImageLoader, para cargar la imagen ocean.jpg de modo que se pueda asignar al objeto BitmapData.

A continuación se muestra el código fuente de la clase ImageLoader (guarde este código en un archivo denominado ImageLoader.as):

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.net.URLRequest;
    public class ImageLoader extends Sprite {
        public var url:String;
        public var bitmap:Bitmap;
    public function ImageLoader(loc:String = null) {
        if (loc != null){
            url = loc;
            loadImage();
        }
    }
    public function loadImage():void{
        if (url != null){
            var loader:Loader = new Loader();
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
            loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, onIoError);

                var req:URLRequest = new URLRequest(url);
                loader.load(req);
            }
        }

    private function onComplete(event:Event):void {
            var loader:Loader = Loader(event.target.loader);
            var info:LoaderInfo = LoaderInfo(loader.contentLoaderInfo);
            this.bitmap = info.content as Bitmap;
            this.dispatchEvent(new Event(Event.COMPLETE));
    }

    private function onIoError(event:IOErrorEvent):void {
            trace("onIoError: " + event);
    }
    }
}
```

Y a continuación se muestra el código ActionScript que utiliza triángulos, mapeado UV y valores T para hacer que la imagen parezca que está rotando y encogiéndose hacia un punto de fuga. Guarde este código en un archivo denominado Spinning3dOcean.as:

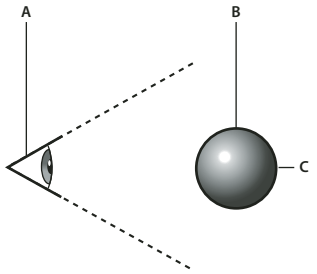
```
package {
    import flash.display.*
    import flash.events.*;
    import flash.utils.getTimer;

    public class Spinning3dOcean extends Sprite {
        // plane vertex coordinates (and t values)
        var x1:Number = -100,y1:Number = -100,z1:Number = 0,t1:Number = 0;
        var x2:Number = 100,y2:Number = -100,z2:Number = 0,t2:Number = 0;
        var x3:Number = 100,y3:Number = 100,z3:Number = 0,t3:Number = 0;
        var x4:Number = -100,y4:Number = 100,z4:Number = 0,t4:Number = 0;
        var focalLength:Number = 200;
        // 2 triangles for 1 plane, indices will always be the same
        var indices:Vector.<int>;

        var container:Sprite;

        var bitmapData:BitmapData; // texture
        var imageLoader:ImageLoader;
        public function Spinning3dOcean():void {
            indices =  new Vector.<int>();
            indices.push(0,1,3, 1,2,3);

            container = new Sprite(); // container to draw triangles in
            container.x = 200;
            container.y = 200;
            addChild(container);

            imageLoader = new ImageLoader("ocean.jpg");
            imageLoader.addEventListener(Event.COMPLETE, onImageLoaded);
        }
        function onImageLoaded(event:Event):void {
            bitmapData = imageLoader.bitmap.bitmapData;
            // animate every frame
            addEventListener(Event.ENTER_FRAME, rotatePlane);
        }
        function rotatePlane(event:Event):void {
            // rotate vertices over time
            var ticker = getTimer()/400;
            z2 = z3 = -(z1 = z4 = 100*Math.sin(ticker));
            x2 = x3 = -(x1 = x4 = 100*Math.cos(ticker));

            // calculate t values
```

```
            t1 = focalLength/(focalLength + z1);
            t2 = focalLength/(focalLength + z2);
            t3 = focalLength/(focalLength + z3);
            t4 = focalLength/(focalLength + z4);

            // determine triangle vertices based on t values
            var vertices:Vector.<Number> = new Vector.<Number>();
            vertices.push(x1*t1,y1*t1, x2*t2,y2*t2, x3*t3,y3*t3, x4*t4,y4*t4);
            // set T values allowing perspective to change
            // as each vertex moves around in z space
            var uvtData:Vector.<Number> = new Vector.<Number>();
            uvtData.push(0,0,t1, 1,0,t2, 1,1,t3, 0,1,t4);

            // draw
            container.graphics.clear();
            container.graphics.beginBitmapFill(bitmapData);
            container.graphics.drawTriangles(vertices, indices, uvtData);
        }
    }
}
```

Para probar este ejemplo, guarde los dos archivos de clase en el mismo directorio que una imagen llamada “ocean.jpg”. Puede ver cómo se transforma el mapa de bits original para que parezca que se aleja en la distancia y gira en un espacio 3D.

Técnica "culling"

La técnica "culling" es el proceso que determina qué superficies de un objeto tridimensional no debe representar el procesador porque están ocultas desde el punto de vista actual. En el espacio 3D, la superficie de la parte “posterior” de un objeto tridimensional queda oculta desde el punto de vista:



La parte posterior de un objeto 3D se oculta desde el punto de vista.
***A.** Punto de vista **B.** Objeto 3D **C.** Parte posterior de un objeto tridimensional*

Los triángulos siempre se representan independientemente de su tamaño, forma o posición. La técnica "culling" garantiza que Flash Player o AIR representen los objetos 3D correctamente. Asimismo, para guardar los ciclos de representación, a veces se desea que el procesador omita algunos triángulos. Pongamos como ejemplo un cubo giratorio en el espacio. En un momento dado, nunca verá más de tres caras del cubo, ya que las caras ocultas estarán frente a la otra dirección del otro lado del cubo. Debido a que estas caras no se van a ver, el procesador no debe dibujarlas. Sin "culling", Flash o AIR representan tanto las caras frontales como las posteriores.

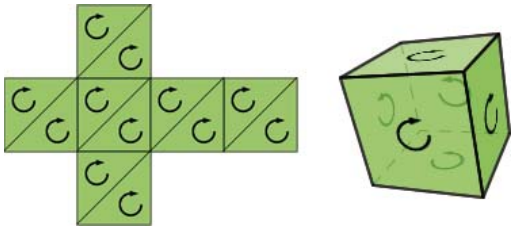
Un cubo tiene caras no visibles desde el punto de vista actual

Por lo tanto, el método `Graphics.drawTriangles()` tiene un cuarto parámetro para establecer un valor de "culling":

```
public function drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null,
uvtData:Vector.<Number> = null, culling:String = "none"):void
```

El parámetro de "culling" es un valor de la clase de enumeración `TriangleCulling`: `TriangleCulling.NONE`, `TriangleCulling.POSITIVE` y `TriangleCulling.NEGATIVE`. Estos valores son dependientes de la dirección del trazado del triángulo que define la superficie del objeto. La API de ActionScript para determinar el valor "culling" da por hecho que todos los triángulos exteriores de una forma 3D se dibujan con la misma dirección de trazado. Cuando una cara del triángulo cambia de dirección, la dirección del trazado también se modifica. En este momento, al triángulo se le puede aplicar el proceso de "culling" (eliminarse de la representación).

Por lo tanto, un valor `TriangleCulling` de `POSITIVE` elimina triángulos con una dirección de trazado positiva (en el sentido de las agujas del reloj). Un valor `TriangleCulling` de `NEGATIVE` elimina triángulos con una dirección de trazado negativa (en sentido contrario a las agujas del reloj). En el caso de un cubo, mientras que las caras frontales tienen una dirección de trazado positiva, las posteriores cuentan con una dirección de trazado negativa:



Cubo "sin ajustar" para mostrar la dirección de trazado. Cuando se "ajusta", se invierte la dirección de trazado de la cara posterior

Para ver cómo funciona la técnica "culling", comience con el ejemplo anterior de "Mapeado UV" en la página 532, establezca el parámetro de "culling" del método`drawTriangles()` en `TriangleCulling.NEGATIVE`:

```
container.graphics.drawTriangles(vertices, indices, uvtData, TriangleCulling.NEGATIVE);
```

Observe que la parte "posterior" de la imagen no se procesa conforme gira el objeto.

Capítulo 24: Trabajo con vídeo

El vídeo de Flash es una de las tecnologías destacadas de Internet. Sin embargo, la presentación tradicional de vídeo (en una pantalla rectangular con una barra de progreso y algunos botones de control debajo) sólo constituye uno de los posibles usos del vídeo. Mediante ActionScript, se obtiene un acceso preciso a la carga, la presentación y la reproducción de vídeo, así como control sobre dichas operaciones.

Fundamentos de la utilización de vídeo

Introducción a la utilización de vídeo

Una de las funciones importantes de Adobe® Flash® Player y Adobe® AIR™ radica en la capacidad de mostrar y manipular información de vídeo con ActionScript del mismo modo que se pueden manipular otros contenidos visuales, como imágenes, animaciones, texto, etc.

Cuando se crea un archivo de Flash Video (FLV) en Adobe Flash CS4 Professional, existe la opción de seleccionar un aspecto que incluya controles de reproducción comunes. No obstante, no hay motivo por el que se deba limitar a las opciones disponibles. Con ActionScript, se obtiene un control preciso de la carga, visualización y la reproducción de vídeo, lo que significa que es posible crear un aspecto de vídeo personalizado o utilizar el vídeo de forma menos tradicional, si se prefiere.

La utilización de vídeo en ActionScript implica trabajar con una combinación de varias clases:

- Clase Video: el cuadro de contenido de vídeo real en el escenario es una instancia de la clase Video. La clase Video es un objeto de visualización, de manera que se puede manipular con las mismas técnicas que se aplican a otros objetos similares, como el ajuste de la posición, la aplicación de transformaciones, la aplicación de filtros y modos de mezcla, etc.
- Clase NetStream: cuando se carga un archivo de vídeo para que se controle mediante ActionScript, una instancia de NetStream representa el origen del contenido de vídeo; en este caso, una trasmisión de datos de vídeo. La utilización de la instancia de NetStream también implica utilizar un objeto NetConnection, que es la conexión al archivo de vídeo, al igual que el túnel con el que se alimentan los datos de vídeo.
- Clase Camera: cuando se trabaja con datos de vídeo desde una cámara conectada al equipo del usuario, una instancia de Camera representa el origen del contenido de vídeo; la cámara del usuario y los datos de vídeo que hace disponibles.

Cuando se carga vídeo externo, se puede cargar el archivo desde un servidor Web estándar para obtener una descarga progresiva, o bien, se puede trabajar con un flujo de vídeo transmitido por un servidor especializado como Flash® Media Server de Adobe.

Tareas comunes relacionadas con el vídeo

En este capítulo se describen las siguientes tareas relacionadas con el vídeo que se realizan frecuentemente:

- Visualizar y controlar vídeo en la pantalla
- Carga de archivos de vídeo externos
- Control de la reproducción de vídeo
- Uso de la pantalla completa

- Gestionar metadatos e información de puntos de referencia en un archivo de vídeo
- Capturar y mostrar entradas de vídeo de la cámara de un usuario

Conceptos y términos importantes

- Punto de referencia: marcador que se puede colocar en un instante de tiempo específico en un archivo de vídeo; por ejemplo, funciona como un marcador para buscar ese instante o proporcionar datos adicionales asociados a dicho momento.
- Codificación: proceso de convertir datos de vídeo en un formato a otro formato; por ejemplo, convertir un vídeo de origen de alta resolución a un formato adecuado para la transmisión por Internet.
- Fotograma: segmento individual de información de vídeo; cada fotograma es como una imagen estática que representa una instantánea de un momento en el tiempo. Si se reproducen fotogramas de forma secuencial a alta velocidad, se crea la ilusión del movimiento.
- Fotograma clave: fotograma de vídeo que contiene información completa del fotograma. El resto de fotogramas que siguen a un fotograma clave sólo contienen información sobre cómo difieren del fotograma clave, en lugar de incluir información relativa al fotograma completo.
- Metadatos: información sobre un archivo de vídeo que se puede incorporar en el archivo de vídeo y recuperarse cuando se haya cargado el vídeo.
- Descarga progresiva: cuando se transmite un archivo de vídeo desde un servidor Web estándar, los datos de vídeo se cargan mediante la descarga progresiva; es decir, la información de vídeo se carga de forma secuencial. Esto presenta la ventaja de que el vídeo puede empezar a reproducirse antes de que se haya descargado todo el archivo; sin embargo, impide poder saltar a una parte del vídeo que no se haya cargado aún.
- Transmisión: como alternativa a la descarga progresiva, se puede utilizar un servidor de vídeo especial para emitir vídeo por Internet mediante la técnica conocida como transmisión de flujo (a veces denominada "transmisión de flujo verdadera"). Con la transmisión de flujo, el ordenador del espectador nunca descarga el vídeo completo a la vez. Para acelerar los tiempos de descarga, el ordenador sólo necesita, en cualquier momento, una parte de la información de vídeo total. Puesto que un servidor especial controla la transmisión del contenido de vídeo, se puede acceder en todo momento a cualquier parte del vídeo, en lugar de tener que esperar a que se descargue antes de acceder a él.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Este capítulo se centra en la utilización de vídeo en ActionScript, por lo que prácticamente todos los listados de código requieren la utilización de un objeto de vídeo, que puede ser un objeto creado y situado en el escenario de Flash, o bien, un objeto creado mediante ActionScript. Para probar un ejemplo es necesario ver el resultado en Flash Player para ver los efectos del código en el vídeo.

La mayoría de los listados de código de ejemplo manipulan un objeto Video sin crear el objeto explícitamente. Para probar los listados de código de este capítulo:

1. Cree un documento de Flash vacío.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Si es necesario, abra el panel Biblioteca.
5. En el menú del panel Biblioteca, elija Nuevo vídeo.

6 En el cuadro de diálogo Propiedades de vídeo, escriba un nombre para el nuevo símbolo de vídeo y elija Vídeo (controlado por ActionScript) en el campo Tipo. Haga clic en Aceptar para crear el símbolo de vídeo.

7 Arrastre una instancia del símbolo de vídeo desde el panel Biblioteca hasta el escenario.

8 Con la instancia de vídeo seleccionada, asígnele un nombre de instancia en el inspector de propiedades. El nombre debe coincidir con el utilizado para la instancia de Video en el listado de código de ejemplo; por ejemplo, si el código manipula un objeto Video denominado`vid`, debe asignar además a la instancia de Stage el nombre `vid`.

9 Ejecute el programa utilizando Control > Probar película.

Verá en pantalla el resultado de la manipulación del vídeo realizada por el código.

Algunos listados de código de ejemplo de este capítulo incluyen además una definición de clase. En esos listados, además de los pasos anteriores, y antes de probar el archivo SWF, hay que crear la clase que se utilizará en el ejemplo. Para crear una clase definida en un listado de código de ejemplo:

1 Asegúrese de que ha guardado el archivo FLA que va a utilizar para las pruebas.

2 En el menú principal, elija Archivo > Nuevo.

3 En el cuadro de diálogo Nuevo documento, en la sección Tipo, elija Archivo ActionScript. Haga clic en Aceptar para crear el nuevo archivo ActionScript.

4 Copie el código de la definición de clase del ejemplo en el documento ActionScript.

5 En el menú principal, seleccione Archivo > Guardar. Guarde el archivo en el mismo directorio que el documento Flash. El nombre del archivo debe coincidir con el nombre de la clase del listado de código. Por ejemplo, si el listado de código define una clase denominada "VideoTest", guarde el archivo ActionScript como "VideoTest.as".

6 Vuelva al documento Flash.

7 Ejecute el programa utilizando Control > Probar película.

Verá el resultado del ejemplo mostrado en pantalla.

En el capítulo “Prueba de los listados de código de ejemplo del capítulo” en la página 36 se explican de forma más detallada otras técnicas para la comprobación de listados de código.

Aspectos básicos de los formatos de vídeo

Además del formato de vídeo Adobe FLV, Flash Player y Adobe AIR admiten audio y vídeo codificados en H.264 y HE-AAC desde formatos de archivo estándar MPEG-4. Estos formatos transmiten vídeo de alta calidad a velocidades de bits más bajas. Los desarrolladores pueden aprovechar herramientas estándar de la industria, incluyendo Adobe Premiere Pro y Adobe After Effects, para crear y proporcionar contenido de vídeo convincente.

Tipo	Formato	Contenedor
Vídeo	H.264	MPEG-4: MP4, M4V, F4V, 3GPP
Vídeo	Archivo FLV	Sorenson Spark
Vídeo	Archivo FLV	ON2 VP6
Audio	AAC+ / HE-AAC / AAC v1 / AAC v2	MPEG-4:MP4, M4V, F4V, 3GPP
Audio	MP3	MP3
Audio	Nellymoser	Archivo FLV
Audio	Speex	Archivo FLV

Compatibilidad de Flash Player y AIR con archivos de vídeo codificado

Flash Player 7 admite archivos FLV que están codificados con códec de vídeo Sorenson™ Spark™. Flash Player 8 admite archivos FLV codificados con el codificador Sorenson Spark u On2 VP6 en Flash Professional 8. El códec de vídeo On2 VP6 admite un canal alfa.

Flash Player 9.0.115.0 y versiones posteriores admiten archivos que proceden del formato contenedor MPEG-4 estándar. Estos archivos incluyen F4V, MP4, M4A, MOV, MP4V, 3GP y 3G2, si contienen vídeo H.264 o audio codificado HE-AAC v2, o ambos. H.264 proporciona mayor calidad de vídeo a velocidades de bits inferiores en comparación con el mismo perfil de codificación en Sorenson o On2. HE-AAC v2 es una extensión de AAC, un formato de audio estándar definido en el estándar de vídeo MPEG-4. HE-AAC v2 utiliza técnicas de réplica de banda espectral (SBR) y estéreo paramétrico (PS) para aumentar la eficacia de la codificación a velocidades de bits bajas.

En la siguiente tabla se incluyen los códecs compatibles. También se muestra el formato de archivo SWF correspondiente y las versiones de Flash Player y AIR necesarias para reproducirlos:

Códec	**Versión del formato de archivo SWF (primera versión de publicación admitida)**	**Flash Player y AIR (primera versión necesaria para reproducción)**
Sorenson Spark	6	Flash Player 6, Flash Lite 3
On2 VP6	6	Flash Player 8, Flash Lite 3. Únicamente Flash Player 8 y versiones posteriores permiten publicar y reproducir vídeo On2 VP6.
H.264 (MPEG-4 Parte 10)	9	Flash Player 9 Update 3, AIR 1.0
ADPCM	6	Flash Player 6, Flash Lite 3
MP3	6	Flash Player 6, Flash Lite 3
AAC (MPEG-4 Parte 3)	9	Flash Player 9 Update 3, AIR 1.0
Speex (audio)	10	Flash Player 10, AIR 1.5
Nellymoser	6	Flash Player 6

Aspectos básicos de los formatos de archivo de vídeo F4V y FLV de Adobe

Adobe proporciona los formatos de archivo de vídeo F4V y FLV para trasmitir contenido a Flash Player y AIR. Para obtener una descripción completa de estos formatos de archivo de vídeo, consulte www.adobe.com/go/video_file_format_es.

Formato de archivo de vídeo F4V

Comenzando con Flash Player actualización 3 (9.0.115.0) y AIR 1.0, Flash Player y AIR admiten el formato de vídeo F4V de Adobe, basado en el formato ISO MP4; los subconjuntos del formato admiten diferentes funciones. Flash Player espera que un archivo F4V válido comience con uno de los siguientes cuadros de nivel superior:

- ftyp

  El cuadro ftyp identifica las funciones que debe admitir un programa para reproducir un formato de archivo concreto.

- moov

  El cuadro moov es realmente el encabezado de un archivo F4V. Contiene uno o varios de los demás cuadros que a su vez incluyen otros cuadros que definen la estructura de los datos F4V. Un archivo F4V debe incluir únicamente un cuadro moov.

- mdat

  Un cuadro mdat contiene la carga útil de datos para el archivo F4V. Un archivo FV contiene únicamente un cuadro mdat. Un cuadro moov también puede estar presente en el archivo, ya que el cuadro mdat no se puede interpretar por sí mismo.

Los archivos F4V admiten enteros multibyte con un orden de bytes bigEndian, en el que el byte más significativo se produce primero, en la dirección más baja.

Formato de archivo de vídeo FLV

El formato del archivo FLV de Adobe contiene datos de audio y vídeo codificados para la publicación con Flash Player. Puede utilizar un codificador, como Adobe Media Encoder orSorenson™ Squeeze, para convertir un archivo de vídeo de QuickTime o Windows Media en un archivo FLV.

Nota: *se pueden crear archivos FLV importando vídeo en Flash y exportándolo como archivo FLV. Se puede utilizar el complemento de exportación de FLV para exportar archivos FLV de aplicaciones de edición de vídeo compatibles. Para cargar archivos FLV desde un servidor web, es necesario registrar la extensión de nombre de archivo y el tipo MIME con el servidor web. Consulte la documentación de su servidor web. El tipo MIME de los archivos FLV es* `video/x-flv`*. Para más información, consulte "Configuración de archivos FLV para alojar en el servidor" en la página 572.*

Para obtener más información sobre archivos FLV, consulte "Temas avanzados para archivos FLV" en la página 572.

Vídeo externo frente a incorporado

La utilización de archivos de vídeo externos ofrece algunas posibilidades que no están disponibles al utilizar vídeo importado:

- Se pueden utilizar clips de vídeo más largos en su aplicación sin que ello ralentice la reproducción. Los archivos de vídeo externos utilizan memoria en caché, lo que significa que los archivos de gran tamaño se almacenan en pequeñas partes y se accede a los mismos de forma dinámica. Por este motivo, los archivos FLV y F4V externos requieren menos memoria que los archivos de vídeo incrustados.
- Un archivo de vídeo externo puede tener una velocidad de fotogramas distinta a la del archivo SWF en el que se reproduce. Por ejemplo, la velocidad de fotogramas del archivo SWF se puede establecer en 30 fotogramas por segundo (fps) y la del vídeo en 21 fps. Esta opción permite un mejor control del vídeo que el vídeo incorporado, para garantizar una reproducción del vídeo sin problemas. Asimismo, permite reproducir archivos de vídeo a distintas velocidades de fotogramas sin necesidad de modificar el contenido del archivo SWF existente.
- Con los archivos de vídeo externos, la reproducción de contenido SWF no se interrumpe mientras que el archivo de vídeo se está cargando. A veces, los archivos de vídeo importados pueden interrumpir la reproducción de un documento para realizar ciertas funciones, como acceder a una unidad de CD-ROM. Los archivos de vídeo pueden realizar funciones independientemente del contenido SWF, sin que la reproducción se vea interrumpida.
- La subtitulación de contenido de vídeo es más fácil con los archivos FLV externos, ya que es posible acceder a los metadatos del vídeo mediante controladores de eventos.

Aspectos básicos de la clase Video

La clase Video permite mostrar un flujo de vídeo en vivo en una aplicación sin incorporarlo al archivo SWF. Se puede capturar y reproducir vídeo en vivo mediante el método `Camera.getCamera()`. También se puede utilizar la clase Video para reproducir archivos de vídeo en HTTP o desde el sistema de archivos local. Hay cuatro formas diferentes de utilizar la clase Video en los proyectos:

- Cargar un archivo de vídeo dinámicamente con las clases NetConnection y NetStream y mostrar el vídeo en un objeto Video.
- Capturar las entradas de la cámara del usuario. Para obtener más información, consulte "Captura de entradas de cámara" en la página 565.
- Utilizar el componente FLVPlayback.

Nota: *las instancias de un objeto Video en el escenario son instancias de la clase Video.*

Aunque la clase Video se encuentra en el paquete flash.media, hereda de la clase flash.display.DisplayObject; por lo tanto, toda la funcionalidad del objeto de visualización (como las transformaciones de matriz y los filtros) también se aplican a las instancias de Video.

Para obtener más información, consulte "Manipulación de objetos de visualización" en la página 300, "Trabajo con la geometría" en la página 351 y "Aplicación de filtros a objetos de visualización" en la página 363.

Carga de archivos de vídeo

La carga de vídeos con las clases NetStream y NetConnection es un proceso de varios pasos:

1. Crear un objeto NetConnection. Si se va a conectar a un archivo de vídeo local o a uno que no utilice un servidor, como Flash Media Server 2 de Adobe, transmita el valor `null` al método `connect()` para reproducir los archivos de vídeo desde una dirección HTTP o una unidad local. Si se conecta a un servidor, defina este parámetro con el URI de la aplicación que contiene el archivo de vídeo en el servidor.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2. Cree un objeto NetStream que adopte un objeto NetConnection como parámetro y especifique el archivo de vídeo que desee cargar. El siguiente fragmento de código conecta un objeto NetStream con la instancia de NetConnection especificada y carga un archivo de vídeo denominado video.mp4 en el mismo directorio que el archivo SWF:

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.mp4");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}
```

3. Cree un nuevo objeto Video y asocie el objeto NetStream creado anteriormente utilizando el método `attachNetStream()` de la clase Video. A continuación, se puede añadir el objeto Video a la lista de visualización con el método `addChild()`, tal como se muestra en el fragmento siguiente:

```
var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Conforme Flash Player ejecuta este código, intenta cargar el archivo de vídeo video.mp4 desde el mismo directorio que su archivo SWF.

Control de la reproducción de vídeo

La clase NetStream ofrece cuatro métodos principales para controlar la reproducción de vídeo:

`pause()`: detiene la reproducción de un flujo de vídeo. Si el vídeo ya está en pausa, la llamada a este método no tendrá ningún efecto.

resume(): reanuda la reproducción de un flujo de vídeo que se ha detenido. Si el vídeo ya se está reproduciendo, la llamada a este método no tendrá ningún efecto.

seek(): busca el fotograma clave más cercano a la ubicación especificada (un desplazamiento, en segundos, desde el comienzo del flujo).

togglePause(): detiene o reanuda la reproducción de un flujo.

Nota: *el método* `stop()` *no está disponible. Para detener un flujo, se debe pausar la reproducción y buscar el principio del flujo de vídeo.*

Nota: *el método* `play()` *no reanuda la reproducción; se utiliza para cargar archivos de vídeo.*

En el ejemplo siguiente se demuestra cómo controlar un vídeo mediante diferentes botones. Para ejecutar el siguiente ejemplo, cree un nuevo documento y añada cuatro instancias de botón al espacio de trabajo (`pauseBtn`, `playBtn`, `stopBtn` y `togglePauseBtn`):

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
    ns.pause();
}
function playHandler(event:MouseEvent):void
{
    ns.resume();
}
function stopHandler(event:MouseEvent):void
{
    // Pause the stream and move the playhead back to
    // the beginning of the stream.
    ns.pause();
    ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
    ns.togglePause();
}
```

Si se hace clic en la instancia de botón `pauseBtn`, el archivo de vídeo quedará en pausa. Si el vídeo ya está en pausa, hacer clic en este botón no tendrá ningún efecto. Si se hace clic en el botón `playBtn`, se reanuda la reproducción de vídeo si ésta estaba en pausa anteriormente; de lo contrario, el botón no tendrá ningún efecto si el vídeo ya estaba en reproducción.

Detección del final de un flujo de vídeo

Para poder detectar el principio y el final de un flujo de vídeo, se debe añadir un detector de eventos a la instancia de NetStream para el evento `netStatus`. En el código siguiente se demuestra cómo detectar varios códigos mediante la reproducción del vídeo:

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
    trace(event.info.code)
}
```

El código anterior genera el resultado siguiente:

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

Los dos códigos que se desean detectar específicamente son "NetStream.Play.Start" y "NetStream.Play.Stop", que señalan el principio y el final de la reproducción del vídeo. El siguiente fragmento utiliza una sentencia switch para filtrar estos dos códigos y rastrear un mensaje:

```
function statusHandler(event:NetStatusEvent):void
{
    switch (event.info.code)
    {
        case "NetStream.Play.Start":
            trace("Start [" + ns.time.toFixed(3) + " seconds]");
            break;
        case "NetStream.Play.Stop":
            trace("Stop [" + ns.time.toFixed(3) + " seconds]");
            break;
    }
}
```

Si se detecta el evento `netStatus` (`NetStatusEvent.NET_STATUS`), se puede crear un reproductor de vídeo que cargue el vídeo siguiente de una lista de reproducción una vez que haya terminado de reproducirse el vídeo actual.

Reproducción de vídeo en modo de pantalla completa

Flash Player y AIR permiten crear una aplicación de pantalla completa para la reproducción de vídeo y admiten la escala de vídeo a pantalla completa.

Para el contenido de AIR que se ejecuta en modo de pantalla completa, el protector de pantalla del sistema y las opciones de ahorro de energía se desactivan durante la reproducción hasta que la entrada de vídeo se detenga o el usuario salga del modo de pantalla completa.

Para obtener información detallada sobre el uso del modo de pantalla completa, consulte "Trabajo con el modo de pantalla completa" en la página 294.

Activación del modo de pantalla completa para Flash Player en un navegador

Antes de que pueda implementar el modo de pantalla completa para Flash Player en un navegador, actívelo mediante la plantilla de publicación para su aplicación. Las plantillas que permiten la pantalla completa incluyen etiquetas `<object>` y `<embed>` que contienen un parámetro `allowFullScreen`. El siguiente ejemplo muestra el parámetro `allowFullScreen` en una etiqueta `<embed>`.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="fullScreen" width="100%" height="100%"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
    ...
    <param name="allowFullScreen" value="true" />
    <embed src="fullScreen.swf" allowFullScreen="true" quality="high" bgcolor="#869ca7"
        width="100%" height="100%" name="fullScreen" align="middle"
        play="true"
        loop="false"
        quality="high"
        allowScriptAccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer">
    </embed>
    ...
</object>
```

En Flash, seleccione Archivo -> Configuración de publicación y en el cuadro de diálogo Configuración de publicación, en la ficha HTML, seleccione la plantilla Sólo Flash - Permitir pantalla completa.

En Flex, asegúrese de que la plantilla HTML incluya las etiquetas `<object>` y `<embed>` que son compatibles con la pantalla completa.

Inicio del modo de pantalla completa

Para el contenido de Flash Player que se ejecuta en un navegador, el modo de pantalla completa se inicia para vídeo como respuesta a un clic de ratón o a una pulsación de tecla. Por ejemplo, el modo de pantalla completa se puede iniciar cuando el usuario hace clic en un botón con la etiqueta de modo de pantalla completa o selecciona un comando Pantalla completa en un menú contextual. Para responder al usuario, añada un detector de eventos al objeto en el que sucede la operación. El siguiente código añade un detector de eventos a un botón en el que el usuario hace clic para introducir el modo de pantalla completa:

```
var fullScreenButton:Button = new Button();
fullScreenButton.label = "Full Screen";
addChild(fullScreenButton);
fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);

function fullScreenButtonHandler(event:MouseEvent)
{
    stage.displayState = StageDisplayState.FULL_SCREEN;

}
```

El código inicia el modo de pantalla completa estableciendo la propiedad `Stage.displayState` en `StageDisplayState.FULL_SCREEN`. Este código escala todo el escenario a modo de pantalla completa con el ajuste de escala de vídeo en proporción al espacio que ocupa en el escenario.

La propiedad `fullScreenSourceRect` permite especificar un área concreta del escenario para realizar un ajuste de escala a pantalla completa. En primer lugar, defina el rectángulo que desee escalar a pantalla completa. Después asígnelo a la propiedad `Stage.fullScreenSourceRect`. Esta versión de la función `fullScreenButtonHandler()` agrega dos líneas adicionales de código que escalan sólo el vídeo a pantalla completa.

```
private function fullScreenButtonHandler(event:MouseEvent)
{
    var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
    stage.fullScreenSourceRect = screenRectangle;
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Aunque en este ejemplo se invoca un controlador de eventos como respuesta a un clic de ratón, la técnica de pasar al modo de pantalla completa es la misma tanto para Flash Player como para AIR. Defina el rectángulo que desee escalar y, a continuación, establezca la propiedad `Stage.displayState`. Para obtener más información, consulte la Referencia del lenguaje y componentes ActionScript 3.0.

El siguiente ejemplo completo agrega código que crea la conexión y el objeto NetStream para el vídeo y comienza a reproducirlo.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import fl.controls.Button;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.FullScreenEvent;
    import flash.geom.Rectangle;

    public class FullScreenVideoExample extends Sprite
    {
        var fullScreenButton:Button = new Button();
        var video:Video = new Video();

        public function FullScreenVideoExample()
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("http://www.helpexamples.com/flash/video/water.flv");

            fullScreenButton.x = 100;
```

```
            fullScreenButton.y = 270;
            fullScreenButton.label = "Full Screen";
            addChild(fullScreenButton);
            fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);
        }

        private function fullScreenButtonHandler(event:MouseEvent)
        {
            var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width,
video.height);
            stage.fullScreenSourceRect = screenRectangle;
            stage.displayState = StageDisplayState.FULL_SCREEN;
        }

        public function onMetaData(infoObject:Object):void
        {
            // stub for callback function
        }
    }
}
```

La función `onMetaData()` es una función callback para administrar metadatos de vídeo, si existen. Una función callback es una función que llama el tiempo de ejecución como respuesta a algún tipo de instancia o evento. En este ejemplo, la función `onMetaData()` es un código auxiliar que cumple con el requisito de proporcionar la función. Para obtener más información, consulte "Escritura de métodos callback para metadatos y puntos de referencia" en la página 551.

Cancelación del modo de pantalla completa

Un usuario puede cancelar el modo de pantalla completa introduciendo uno de los métodos abreviados de teclado como, por ejemplo, la tecla Esc. Para cancelar el modo de pantalla completa en ActionScript, establezca la propiedad `Stage.diplayState` en `StageDisplayState.NORMAL`. El código del siguiente ejemplo cancela el modo de pantalla completa cuando se produce el evento `netStatus` de NetStream.Play.Stop.

```
videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

private function netStatusHandler(event:NetStatusEvent)
{
    if(event.info.code == "NetStream.Play.Stop")
        stage.displayState = StageDisplayState.NORMAL;
}
```

Aceleración de hardware a pantalla completa

Cuando se establece la propiedad `Stage.fullScreenSourceRect` para escalar un segmento rectangular del escenario en modo de pantalla completa, Flash Player o AIR utilizan la aceleración de hardware, si está disponible y habilitada. El tiempo de ejecución utiliza el adaptador de vídeo del equipo para agilizar el ajuste de escala del vídeo, o una parte del escenario, al tamaño de pantalla competa.

Para obtener más información sobre la aceleración de hardware a pantalla completa, consulte "Trabajo con el modo de pantalla completa" en la página 294.

Transmisión de archivos de vídeo

Para transmitir archivos de Flash Media Server, se pueden utilizar las clases NetConnection y NetStream para conectarse a una instancia de servidor remoto y reproducir un flujo especificado. Para especificar un servidor RTMP (Real-Time Messaging Protocol), la URL de RTMP deseada como, por ejemplo, "rtmp://localhost/appName/appInstance", se transmite al método `NetConnection.connect()` en lugar de transmitir un valor `null`. Para reproducir un flujo grabado o en directo determinado desde el servidor Flash Media Server especificado, se transmite un nombre identificativo para los datos en directo publicados por `NetStream.publish()`, o bien, un nombre de archivo grabado para reproducción al método `NetStream.play()`. Para más información, consulte la documentación de Flash Media Server.

Aspectos básicos de los puntos de referencia

Puede incorporar puntos de referencia en un archivo de vídeo F4V o FLV de Adobe durante la codificación. Los puntos de referencia siempre se han incorporado a las películas para proporcionar al proyectista una señal visual que indicara que la cinta estaba llegando a su fin. En los formatos de vídeo F4V y FLV de Adobe, un punto de referencia permite activar una o varias acciones en la aplicación en el momento en que se producen en el flujo de vídeo.

Se pueden utilizar diferentes tipos de puntos de referencia con Flash Video. ActionScript permite interactuar con puntos de referencia que se incorporen en un archivo de vídeo al crearlo.

- Puntos de referencia de navegación: estos puntos se incorporan al flujo de vídeo y al paquete de metadatos al codificar el archivo de vídeo. Los puntos de referencia de navegación se utilizan para permitir a los usuarios buscar una parte especificada de un archivo.
- Puntos de referencia de evento: los puntos de referencia de evento se incorporan al flujo de vídeo y al paquete de metadatos FLV al codificar el archivo de vídeo. Se puede escribir código para controlar eventos que se activan en puntos especificados durante la reproducción de vídeo.
- Puntos de referencia de ActionScript: se trata de puntos referencia que sólo están disponibles en el componente FLVPlayback de Flash. Los puntos de referencia de ActionScript son puntos externos que se crean y se accede a ellos con el código de ActionScript. Se puede escribir código para activar estos puntos de referencia en relación con la reproducción del vídeo. Estos puntos de referencia son menos precisos que los incorporados (hasta una décima de segundo), ya que el reproductor de vídeo realiza un seguimiento de los mismos de forma independiente. Si se va a crear una aplicación en la que se desea que los usuarios naveguen a un punto de referencia, se deben crear e incorporar puntos de referencia al codificar el archivo en lugar de utilizar puntos de referencia de ActionScript. Se recomienda incorporar los puntos de referencia en el archivo FLV, ya que resultan más precisos.

Los puntos de referencia de navegación crean un fotograma clave en una ubicación especificada, por lo que se puede utilizar código para desplazar la cabeza lectora del reproductor de vídeo a dicha ubicación. Se pueden establecer puntos determinados en un archivo de vídeo donde se desee que busquen los usuarios. Por ejemplo, si el vídeo incluyera varios capítulos o segmentos, se podría controlar mediante la incorporación de puntos de referencia de navegación en el archivo de vídeo.

Para más información sobre la codificación de archivos de vídeo de Adobe con puntos de referencia, consulte "Incorporación de puntos de referencia" *en Utilización de Flash*.

Se puede acceder a parámetros de punto de referencia al escribir código ActionScript. Los parámetros de punto de referencia constituyen una parte del objeto de evento recibido mediante el controlador callback.

Para activar determinadas acciones en el código cuando un archivo FLV alcanza un punto de referencia específico, se utiliza el controlador de eventos `NetStream.onCuePoint`.

Para sincronizar una acción para un punto de referencia en un archivo de vídeo F4V, debe recuperar los datos del punto desde las funciones callback `onMetaData()` o `onXMPData()` y activar el punto de referencia utilizando la clase Timer en ActionScript 3.0. Para obtener más información sobre los puntos de referencia F4V, consulte "Utilización de onXMPData()" en la página 562.

Para obtener más información sobre la administración de puntos de referencia y metadatos, consulte "Escritura de métodos callback para metadatos y puntos de referencia" en la página 551.

Escritura de métodos callback para metadatos y puntos de referencia

Puede activar acciones en su aplicación cuando el reproductor reciba metadatos específicos o cuando se alcancen puntos de referencia concretos. Cuando se produzcan estos eventos, debe utilizar métodos callback específicos como controladores de eventos. La clase NetStream especifica los siguientes eventos de metadatos que pueden suceder durante la reproducción: `onCuePoint` (sólo archivos FLV), `onImageData`, `onMetaData`, `onPlayStatus`, `onTextData` y `onXMPData`.

Se deben escribir métodos callback para estos controladores; de lo contrario, Flash Player podría generar errores. Por ejemplo, en el código siguiente se reproduce un archivo FLV denominado video.flv en la misma carpeta donde se encuentra el archivo SWF:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

El código anterior carga un archivo de vídeo local llamado video.flv y detecta la distribución de `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`). Este evento se distribuye cuando se genera una excepción desde el código asincrónico nativo. En este caso, se distribuye cuando un el archivo de vídeo contiene metadatos o información de punto de referencia y no se han definido los detectores apropiados. El código anterior controla el evento `asyncError` y omite el error si no interesan los metadatos o la información de punto de referencia del archivo de vídeo. Si tuviera un archivo FLV con metadatos y varios puntos de referencia, la función `trace()` mostraría los siguientes mensajes de error:

```
Error #2095: flash.net.NetStream was unable to invoke callback onMetaData.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
```

Los errores se producen porque el objeto NetStream no ha podido encontrar un método callback `onMetaData` o `onCuePoint`. Hay varias maneras de definir estos métodos callback en las aplicaciones:

Definición de la propiedad client del objeto NetStream en Object

Al establecer la propiedad `client` en Object o una subclase de NetStream, se pueden redirigir los métodos callback `onMetaData` y `onCuePoint`, o bien, omitirlos completamente. En el ejemplo siguiente se demuestra cómo se puede utilizar una clase Object vacía para omitir los métodos callback sin detectar el evento `asyncError`.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Si se desea detectar los métodos callback `onMetaData` o `onCuePoint`, es necesario definir métodos para controlarlos, tal y como se muestra en el siguiente fragmento de código:

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
```

El código anterior detecta el método callback `onMetaData` y llama al método `metaDataHandler()`, que rastrea una cadena. Si Flash Player ha detectado un punto de referencia, no se generarán errores aunque se defina el método callback `onCuePoint`.

Creación de una clase personalizada y definición de métodos para controlar los métodos callback

En el código siguiente se establece la propiedad `client` del objeto NetStream en una clase personalizada, CustomClient, que define controladores para los métodos callback:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

La clase CustomClient se presenta del siguiente modo:

```
package
{
    public class CustomClient
    {
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
    }
}
```

La clase CustomClient define un controlador para el controlador callback `onMetaData`. Si se ha detectado un punto de referencia y se ha llamado al controlador callback `onCuePoint`, se distribuirá un evento `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) indicando que flash.net.NetStream no ha podido invocar la función callback onCuePoint. Para evitar este error, se debe definir un método callback `onCuePoint` en la clase CustomClient, o bien definir un controlador de eventos para el evento `asyncError`.

Ampliación de la clase NetStream y adición de métodos para controlar los métodos callback

El código siguiente crea una instancia de la clase CustomNetStream, que se define en un listado de código posterior:

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

En la lista de códigos siguiente se define la clase CustomNetStream que amplía la clase NetStream y controla la creación del objeto NetConnection necesario y los métodos de controlador callback `onMetaData` y `onCuePoint`:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function CustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
        public function onCuePoint(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

Si se desea cambiar el nombre de los métodos `onMetaData()` y `onCuePoint()` en la clase CustomNetStream, se podría utilizar el código siguiente:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public var onMetaData:Function;
        public var onCuePoint:Function;
        public function CustomNetStream()
        {
            onMetaData = metaDataHandler;
            onCuePoint = cuePointHandler;
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        private function metaDataHandler(infoObject:Object):void
        {
            trace("metadata");
        }
        private function cuePointHandler(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

Ampliación y dinamización de la clase NetStream

Se puede ampliar la clase NetStream y hacer que la subclase sea dinámica, de manera que los controladores callback `onCuePoint` y `onMetaData` puedan añadirse dinámicamente. Esto se ilustra en la lista siguiente:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

La clase DynamicCustomNetStream es de esta forma:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public dynamic class DynamicCustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function DynamicCustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
    }
}
```

Incluso sin controladores para los controladores callback `onMetaData` y `onCuePoint`, no se generan errores, ya que la clase DynamicCustomNetStream es dinámica. Si se desea definir métodos para los controladores callback `onMetaData` y `onCuePoint`, se puede utilizar el código siguiente:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
    trace("cue point");
}
```

Establecimiento de la propiedad client del objeto NetStream en this

Si la propiedad `client` se establece en `this`, la aplicación busca en el ámbito actual los métodos `onMetaData()` y `onCuePoint()`. Esto se puede observar en el ejemplo siguiente:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Si se llama a los controladores callback `onMetaData` o `onCuePoint` y no hay métodos para controlar la función callback, no se generan errores. Para controlar estos controladores callback, hay que crear métodos `onMetaData()` y `onCuePoint()` en el código, tal como se muestra en el siguiente fragmento de código:

```
function onMetaData(infoObject:Object):void
{
    trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
    trace("cue point");
}
```

Utilización de puntos de referencia y metadatos

Los métodos callback NetStream se utilizan para capturar y procesar eventos de metadatos y puntos de referencia conforme se reproduce el vídeo.

Utilización de puntos de referencia

En la siguiente tabla se describen los métodos callback que se pueden utilizar para capturar puntos de referencia F4V y FLV en Flash Player y AIR.

Tiempo de ejecución	F4V	FLV
Flash Player 9/ AIR1.0		OnCuePoint
		OnMetaData
Flash Player 10		OnCuePoint
	OnMetaData	OnMetaData
	OnXMPData	OnXMPData

En el siguiente ejemplo se utiliza un sencillo bucle `for..in` para repetir todas las propiedades del parámetro `infoObject` que recibe la función `onCuePoint()`. Llama a la función trace() para que aparezca un mensaje cuando reciba los datos de punto de referencia:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onCuePoint(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

Aparece el siguiente resultado:

```
parameters:
name: point1
time: 0.418
type: navigation
```

Este código utiliza una de las distintas técnicas para configurar el objeto en el que se ejecuta el método callback. También puede utilizar otras técnicas; para obtener más información, consulte "Escritura de métodos callback para metadatos y puntos de referencia" en la página 551.

Utilización de metadatos de vídeo

Las funciones `OnMetaData()` y `OnXMPData()` se pueden emplear para acceder a la información de metadatos en su archivo de vídeo, incluyendo puntos de referencia.

Utilización de OnMetaData()

Los metadatos incluyen información sobre el archivo de vídeo, como la duración, anchura, altura y velocidad de fotogramas. La información de metadatos que se añade al archivo de vídeo depende del software que se utilice para codificar el archivo de vídeo.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

El código anterior genera resultados como los siguientes:

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

Si el vídeo no tiene sonido, la información de metadatos relativa al audio (como `audiodatarate`) devuelve `undefined`, ya que no se ha añadido información de audio a los metadatos durante la codificación.

En el código anterior, no se muestra la información de punto de referencia. Para ver los metadatos del punto de referencia, se puede utilizar la siguiente función que muestra de forma sucesiva los elementos en una clase Object:

```
function traceObject(obj:Object, indent:uint = 0):void
{
    var indentString:String = "";
    var i:uint;
    var prop:String;
    var val:*;
    for (i = 0; i < indent; i++)
    {
        indentString += "\t";
    }
    for (prop in obj)
    {
        val = obj[prop];
        if (typeof(val) == "object")
        {
            trace(indentString + " " + prop + ": [Object]");
            traceObject(val, indent + 1);
        }
        else
        {
            trace(indentString + " " + prop + ": " + val);
        }
    }
}
```

Mediante el fragmento de código anterior para rastrear el parámetro `infoObject` del método `onMetaData()`, se crea la salida siguiente:

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
height: 213
framerate: 15
cuePoints: [Object]
    0: [Object]
        parameters: [Object]
            lights: beginning
        name: point1
        time: 0.418
        type: navigation
    1: [Object]
        parameters: [Object]
            lights: middle
        name: point2
        time: 7.748
        type: navigation
    2: [Object]
        parameters: [Object]
            lights: end
        name: point3
        time: 16.02
        type: navigation
```

El siguiente ejemplo muestra los metadatos para un vídeo MP4. Se da por hecho que existe un objeto llamado `metaDataOut`, en el que se escriben los metadatos.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.events.NetStatusEvent;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class onMetaDataExample extends Sprite
    {
        var video:Video = new Video();

        public function onMetaDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);
```

```
            video.x = 185;
            video.y = 5;

            video.attachNetStream(videoStream);

            videoStream.play("video.mp4");

            videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
        }

        public function onMetaData(infoObject:Object):void
        {
            for(var propName:String in infoObject)
            {
                metaDataOut.appendText(propName + "=" + infoObject[propName] + "\n");
            }
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if(event.info.code == "NetStream.Play.Stop")
                stage.displayState = StageDisplayState.NORMAL;
        }
    }
}
```

La función `onMetaData()` generó los siguientes resultados para este vídeo:

```
moovposition=731965
height=352
avclevel=21
videocodecid=avc1
duration=2.36
width=704
videoframerate=25
avcprofile=88
trackinfo=[object Object]
```

Utilización del objeto information

La siguiente tabla muestra los posibles valores para los metadatos de vídeo que se transmiten a la función callback `onMetaData()` en el objeto que reciben:

Parámetro	Descripción
aacaot	Tipo de objeto de audio AAC; se admiten 0, 1 ó 2.
avclevel	Número de nivel AVC IDC como, por ejemplo, 10, 11, 20, 21, etc.
avcprofile	Número de perfil AVC como, por ejemplo, 55, 77, 100, etc.
audiocodecid	Cadena que indica el códec de audio (técnica de codificación/descodificación) utilizado; por ejemplo, ".Mp3" o "mp4a"
audiodatarate	Número que indica la velocidad a la que se ha codificado el audio, expresada en kilobytes por segundo.
audiodelay	Número que indica el tiempo del archivo FLV correspondiente al "tiempo 0" del archivo FLV original. Es necesario demorar ligeramente el contenido del vídeo para sincronizarlo correctamente con el audio.

Parámetro	Descripción
canSeekToEnd	Un valor booleano que es `true` si el archivo FLV se codifica con un fotograma clave en el último fotograma, lo que permite buscar hasta el final de un clip de película de descarga progresiva. Es `false` si el archivo FLV no se codifica con un fotograma clave en el último fotograma.
cuePoints	Conjunto de objetos, uno en cada punto de referencia incorporado en el archivo FLV. El valor es undefined si el archivo FLV no contiene ningún punto de referencia. Cada objeto tiene las siguientes propiedades: • `type`: cadena que especifica el tipo de punto de referencia como "navigation" o "event". • `name`: cadena que indica el nombre del punto de referencia. • `time`: número que indica el tiempo del punto de referencia en segundos, con una precisión de tres decimales (milisegundos). • `parameters`: objeto opcional que tiene pares nombre-valor designados por el usuario durante la creación de los puntos de referencia.
duration	Número que especifica la duración del archivo de vídeo, en segundos.
framerate	Número que especifica la velocidad de fotogramas del archivo FLV.
height	Número que especifica la altura del archivo FLV, en píxeles.
seekpoints	Conjunto que incluye los fotogramas clave disponibles como marcas horarias en milisegundos. Opcional.
etiquetas	Conjunto de pares clave-valor que representa la información en el átomo “ilst”, que es el equivalente de las etiquetas ID3 para archivos MP4. iTunes utiliza estas etiquetas. Se puede utilizar para mostrar ilustraciones, si está disponible.
trackinfo	Objeto que proporciona información sobre todas las pistas del archivo MP4, incluyendo su ID de descripción de ejemplo.
videocodecid	Cadena que es la versión de códec que se utilizó para codificar el vídeo; por ejemplo, “avc1” o “VP6F”.
videodatarate	Número que especifica la velocidad de datos de vídeo del archivo FLV.
videoframerate	Velocidad de fotogramas del vídeo MP4.
width	Número que especifica la anchura del archivo FLV, en píxeles.

La tabla siguiente muestra los posibles valores del parámetro `videocodecid`:

videocodecid	Nombre de códec
2	Sorenson H.263
3	Screen video (sólo en SWF versión 7 y posterior)
4	VP6 (sólo en SWF versión 8 y posterior)
5	Vídeo VP6 con canal alfa (sólo en SWF versión 8 y posterior)

La tabla siguiente muestra los posibles valores del parámetro `audiocodecid`:

audiocodecid	Nombre de códec
0	uncompressed
1	ADPCM
2	MP3
4	Nellymoser @ 16 kHz mono

audiocodecid	Nombre de códec
5	Nellymoser, 8kHz mono
6	Nellymoser
10	AAC
11	Speex

Utilización de onXMPData()

La función callback `onXMPData()` recibe información específica de Adobe Extensible Metadata Platform (XMP) incorporada en el archivo de vídeo Adobe F4V o FLV. Los metadatos XMP incluyen puntos de referencia, así como otros metadatos de vídeo. La compatibilidad con metadatos XMP se incorpora a partir de Flash Player 10 y Adobe AIR 1.5, y se admite en versiones posteriores de Flash Player y AIR.

En el siguiente ejemplo se procesan datos del punto de referencia en los metadatos XMP:

```
package
{
    import flash.display.*;
    import flash.net.*;
    import flash.events.NetStatusEvent;
    import flash.media.Video;

    public class onXMPDataExample extends Sprite
    {
        public function onXMPDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;
            var video:Video = new Video();

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("video.f4v");
        }

        public function onMetaData(info:Object):void {
            trace("onMetaData fired");
        }

        public function onXMPData(infoObject:Object):void
        {
            trace("onXMPData Fired\n");
            //trace("raw XMP =\n");
            //trace(infoObject.data);
            var cuePoints:Array = new Array();
            var cuePoint:Object;
            var strFrameRate:String;
            var nTracksFrameRate:Number;
            var strTracks:String = "";
```

```
            var onXMPXML = new XML(infoObject.data);
            // Set up namespaces to make referencing easier
            var xmpDM:Namespace = new Namespace("http://ns.adobe.com/xmp/1.0/DynamicMedia/");
            var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
            for each (var it:XML in onXMPXML..xmpDM::Tracks)
            {
                 var strTrackName:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::trackName;
                 var strFrameRateXML:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::frameRate;
                 strFrameRate = strFrameRateXML.substr(1,strFrameRateXML.length);

                 nTracksFrameRate = Number(strFrameRate);

                 strTracks += it;
            }
            var onXMPTracksXML:XML = new XML(strTracks);
            var strCuepoints:String = "";
            for each (var item:XML in onXMPTracksXML..xmpDM::markers)
            {
                strCuepoints += item;
            }
            trace(strCuepoints);
        }
    }
}
```

Para un breve archivo de vídeo denominado startrekintro.f4v, este ejemplo produce las siguientes líneas de trazado. Las líneas muestran los datos del punto de referencia para los puntos de referencia de evento y navegación en los metadatos XMP:

```
onMetaData fired
onXMPData Fired

<xmpDM:markers xmlns:xmp="http://ns.adobe.com/xap/1.0/"
xmlns:xmpDM="http://ns.adobe.com/xmp/1.0/DynamicMedia/"
xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:x="adobe:ns:meta/">
  <rdf:Seq>
    <rdf:li>
      <rdf:Description xmpDM:startTime="7695905817600" xmpDM:name="Title1"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Navigation">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="Title" xmpDM:value="Star Trek"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
    <rdf:li>
      <rdf:Description xmpDM:startTime="10289459980800" xmpDM:name="Title2"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Event">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="William Shatner" xmpDM:value="First Star"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Light Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
  </rdf:Seq>
</xmpDM:markers>
onMetaData fired
```

Nota: *en los datos XMP, el tiempo se almacena como ticks DVA en lugar de en segundos. Para calcular la hora del punto de referencia, divida la hora de inicio entre la velocidad de fotogramas. Por ejemplo, la hora de inicio de 7695905817600 dividida entre una velocidad de fotogramas de 254016000000 es igual a 30:30.*

Para ver los metadatos XMP completos sin formato, lo que incluye la velocidad de fotogramas, elimine los identificadores de comentarios (//'s) que preceden a la segunda y tercera sentencia `trace()` al principio de la función `onXMPData()`.

Para obtener más información sobre XMP, consulte:

- http://partners.adobe.com/public/developer/xmp/topic.html
- http://www.adobe.com/devnet/xmp/

Uso de metadatos de imagen

El evento `onImageData` envía los datos de imagen como un conjunto de bytes a través de un canal de datos AMF0. Los datos se pueden presentar en formatos JPEG, PNG o GIF. Defina un método callback `onImageData()` para procesar esta información, del mismo modo que definiría métodos callback para `onCuePoint` y `onMetaData`. El siguiente ejemplo muestra y accede a los datos de imagen utilizando el método callback `onImageData()`:

```
public function onImageData(imageData:Object):void
{
    // display track number
    trace(imageData.trackid);
    var loader:Loader = new Loader();
    //imageData.data is a ByteArray object
    loader.loadBytes(imageData.data);
    addChild(loader);
}
```

Uso de metadatos de texto

El evento `onTextData` envía datos de texto mediante un canal de datos AMF0. Los datos de texto presentan un formato UTF-8 y contienen información adicional sobre el formato, en función de la especificación de texto temporizado 3GP. Esta especificación define un formato de subtítulo estandarizado. Defina un método callback `onTextData()` para procesar esta información, del mismo modo que definiría métodos callback para `onCuePoint` o `onMetaData`. En el siguiente ejemplo, el método `onTextData()` muestra el número de ID de pista y el texto de pista correspondiente.

```
public function onTextData(textData:Object):void
{
    // display the track number
    trace(textData.trackid);
    // displays the text, which can be a null string, indicating old text
    // that should be erased
    trace(textData.text);
}
```

Captura de entradas de cámara

Además de los archivos de vídeo externos, una cámara conectada al ordenador del usuario puede servir de origen de datos de vídeo, que a su vez se pueden mostrar y manipular con ActionScript. La clase Camera es el mecanismo incorporado en ActionScript para utilizar una cámara de ordenador.

Aspectos básicos de la clase Camera

El objeto Camera permite conectar la cámara local del usuario y difundir vídeo localmente (de vuelta al usuario) o de forma remota a un servidor (como Flash Media Server).

Mediante la clase Camera, se pueden utilizar los siguientes tipos de información sobre la cámara del usuario:

- Las cámaras instaladas en el ordenador del usuario que están disponibles para Flash Player
- Si la cámara está instalada
- Si Flash Player tiene permiso para acceder a la cámara del usuario
- La cámara que está activa en ese momento
- La anchura y la altura del vídeo que se captura

La clase Camera incluye varios métodos y propiedades útiles para utilizar objetos Camera. Por ejemplo, la propiedad `Camera.names` estática contiene un conjunto de nombres de cámara instalados en ese momento en el ordenador del usuario. Asimismo, se puede utilizar la propiedad `name` para mostrar el nombre de la cámara activa en ese momento.

Visualización del contenido de la cámara en pantalla

La conexión a una cámara puede requerir menos código que utilizar las clases NetConnection y NetStream para cargar un vídeo. Asimismo, la clase Camera puede plantear problemas rápidamente, ya que se necesita el permiso de un usuario para que Flash Player se conecte a su cámara antes de que poder acceder a ella.

En el código siguiente se muestra cómo se puede utilizar la clase Camera para conectarse a la cámara local de un usuario:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Nota: *la clase Camera no tiene un método constructor. Para crear una nueva instancia de Camera, se utiliza el método* `Camera.getCamera()` *estático.*

Diseño de la aplicación de cámara

Al programar una aplicación que se conecta a la cámara de un usuario, el código debe hacer lo siguiente:

- Comprobar si el usuario tiene una cámara instalada en ese momento.
- Sólo en Flash Player, compruebe si el usuario ha permitido el acceso a la cámara de forma explícita. Por motivos de seguridad, el reproductor muestra el cuadro de diálogo Configuración de Flash Player, en el que el usuario permite o deniega el acceso a su cámara. Esto impide que Flash Player se conecte a la cámara de un usuario y difunda un flujo de vídeo sin su permiso. Si el usuario permite el acceso haciendo clic en el botón correspondiente, la aplicación puede conectarse a la cámara. De lo contrario, la aplicación no podrá acceder a dicha cámara. Las aplicaciones siempre deben controlar ambas situaciones adecuadamente.

Conexión a la cámara de un usuario

El primer paso de la conexión a la cámara de un usuario consiste en crear una nueva instancia de cámara; se crea una variable de tipo Camera, que se inicializa al valor devuelto del método `Camera.getCamera()` estático.

El siguiente paso es crear un nuevo objeto Video y asociarle el objeto Camera.

El tercer paso consiste en añadir el objeto Video a la lista de visualización. Los pasos 2 y 3 son necesarios, ya que la clase Camera no amplía la clase DisplayObject y no se puede añadir directamente a la lista. Para mostrar el vídeo capturado de la cámara, se crea un nuevo objeto Video y se llama al método `attachCamera()`.

En el código siguiente se muestran estos tres pasos:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Se debe tener en cuenta que si un usuario no dispone de una cámara instalada, la aplicación no mostrará nada.

En situaciones reales, deben llevarse a cabo pasos adicionales para la aplicación. Consulte "Comprobación de que las cámaras están instaladas" en la página 567 y "Detección de permisos para el acceso a una cámara" en la página 567 para obtener más información.

Comprobación de que las cámaras están instaladas

Antes de intentar utilizar métodos o propiedades en una instancia de cámara, tal vez se desee comprobar que el usuario tiene una cámara instalada. Existen dos maneras de comprobar si el usuario tiene una cámara instalada:

- Comprobar la propiedad estática `Camera.names` que contiene un conjunto de nombres de cámara disponibles. Normalmente, este conjunto tiene una cadena o ninguna, puesto que la mayoría de usuarios no disponen de más de una cámara instalada a la vez. En el código siguiente se muestra cómo se puede comprobar la propiedad `Camera.names` para ver si el usuario tiene cámaras disponibles:

```
if (Camera.names.length > 0)
{
    trace("User has at least one camera installed.");
    var cam:Camera = Camera.getCamera(); // Get default camera.
}
else
{
    trace("User has no cameras installed.");
}
```

- Compruebe el valor devuelto del método estático `Camera.getCamera()`. Si no hay cámaras instaladas o disponibles, este método devuelve `null`; de lo contrario, devuelve una referencia a un objeto Camera. En el código siguiente se muestra cómo se puede comprobar el método `Camera.getCamera()` para ver si el usuario tiene cámaras disponibles:

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
    trace("User has no cameras installed.");
}
else
{
    trace("User has at least 1 camera installed.");
}
```

Debido a que la clase Camera no amplía la clase DisplayObject, no se puede añadir directamente a la lista de visualización utilizando el método `addChild()`. Para mostrar el vídeo capturado de la cámara, se debe crear un nuevo objeto Video y llamar al método `attachCamera()` en la instancia de Video.

Este fragmento de código muestra cómo se puede conectar la cámara, si se dispone de ella; de lo contrario, la aplicación no mostrará nada.

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
```

Detección de permisos para el acceso a una cámara

En el entorno limitado de la aplicación AIR, la aplicación puede acceder a cualquier cámara sin el permiso del usuario.

Antes de que Flash Player pueda mostrar la salida de una cámara, el usuario debe permitir explícitamente a la aplicación que acceda a la cámara. Cuando se llama al método `attachCamera()`, Flash Player muestra el cuadro de diálogo Configuración de Flash Player, que pregunta al usuario si desea que Flash Player acceda a la cámara y al micrófono. Si el usuario hace clic en el botón Permitir, Flash Player muestra la salida de la cámara en la instancia de Video en el escenario. Si el usuario hace clic en el botón de rechazo, Flash Player no puede conectarse a la cámara y el objeto Video no muestra nada.

Si se desea saber si el usuario ha permitido o rechazado el acceso a la cámara, se puede detectar el evento `status` (`StatusEvent.STATUS`) de la cámara, tal y como se muestra en el código siguiente:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
    // This event gets dispatched when the user clicks the "Allow" or "Deny"
    // button in the Flash Player Settings dialog box.
    trace(event.code); // "Camera.Muted" or "Camera.Unmuted"
}
```

La función `statusHandler()` se llama cuando el usuario hace clic en los botones para permitir o denegar el acceso. Se puede detectar el botón en el que ha hecho clic el usuario con uno de estos dos métodos:

- El parámetro `event` de la función `statusHandler()` contiene una propiedad de código que incluye la cadena "Camera.Muted" o "Camera.Unmuted". Si el valor es "Camera.Muted", el usuario ha hecho clic en el botón para denegar el acceso y Flash Player no puede acceder a la cámara. Se puede ver un ejemplo de esto en el fragmento siguiente:

  ```
  function statusHandler(event:StatusEvent):void
  {
      switch (event.code)
      {
          case "Camera.Muted":
              trace("User clicked Deny.");
              break;
          case "Camera.Unmuted":
              trace("User clicked Accept.");
              break;
      }
  }
  ```

- La clase Camera contiene una propiedad de sólo lectura denominada `muted`, que especifica si el usuario ha denegado el acceso a la cámara (`true`) o lo ha permitido (`false`) en el panel Privacidad de Flash Player. Se puede ver un ejemplo de esto en el fragmento siguiente:

```
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("User clicked Deny.");
    }
    else
    {
        trace("User clicked Accept.");
    }
}
```

Al buscar el evento de estado que se va a distribuir, se puede escribir código que controle la aceptación o el rechazo del acceso a la cámara por parte del usuario y llevar a cambio la limpieza correspondiente. Por ejemplo, si el usuario hace clic en el botón para denegar el acceso, se puede mostrar un mensaje al usuario que indique que debe hacer clic en el botón para permitir el acceso si desea participar en un chat de vídeo. Asimismo, se puede asegurar de que el objeto Video esté eliminado de la lista de visualización para liberar recursos de sistema.

Aumento de la calidad de vídeo

De forma predeterminada, las nuevas instancias de la clase Video son de 320 píxeles de ancho x 240 píxeles de alto. Para aumentar la calidad de vídeo, siempre debe asegurarse de que el objeto Video coincide con las mismas dimensiones que el vídeo que devuelve el objeto Camera. Se puede obtener la anchura y la altura del objeto Camera mediante las propiedades `width` y `height` de la clase Camera. Asimismo, se pueden establecer las propiedades `width` y `height` del objeto Video para que se correspondan con las dimensiones de los objetos Camera. También es posible pasar la anchura y la altura al método constructor de la clase Video, tal como se muestra en el fragmento siguiente:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video(cam.width, cam.height);
    vid.attachCamera(cam);
    addChild(vid);
}
```

Puesto que el método `getCamera()` devuelve una referencia a un objeto Camera (o `null` si no hay cámaras disponibles), se puede acceder a los métodos y las propiedades de la cámara aunque el usuario deniegue el acceso a ella. Esto permite establecer el tamaño de la instancia de vídeo con la altura y la anchura nativas de la cámara.

```
var vid:Video;
var cam:Camera = Camera.getCamera();

if (cam == null)
{
    trace("Unable to locate available cameras.");
}
else
{
    trace("Found camera: " + cam.name);
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("Unable to connect to active camera.");
    }
    else
    {
        // Resize Video object to match camera settings and
        // add the video to the display list.
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
    }
    // Remove the status event listener.
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

Para obtener información sobre el modo de pantalla completa, consulte la sección sobre este tema en "Configuración de las propiedades de Stage" en la página 292.

Control de las condiciones de reproducción

La clase Camera contiene varias propiedades que permiten controlar el estado actual del objeto Camera. Por ejemplo, el código siguiente muestra varias de las propiedades de la cámara mediante un objeto Timer y una instancia de campo de texto en la lista de visualización:

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (!cam.muted)
    {
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
        t.start();
    }
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
function timerHandler(event:TimerEvent):void
{
    tf.text = "";
    tf.appendText("activityLevel: " + cam.activityLevel + "\n");
    tf.appendText("bandwidth: " + cam.bandwidth + "\n");
    tf.appendText("currentFPS: " + cam.currentFPS + "\n");
    tf.appendText("fps: " + cam.fps + "\n");
    tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
    tf.appendText("loopback: " + cam.loopback + "\n");
    tf.appendText("motionLevel: " + cam.motionLevel + "\n");
    tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
    tf.appendText("quality: " + cam.quality + "\n");
}
```

Cada 1/10 de segundo (100 milisegundos) se distribuye el evento `timer` del objeto Timer y la función `timerHandler()` actualiza el campo de texto de la lista de visualización.

Envío de vídeo a un servidor

Si se desean crear aplicaciones más complejas con objetos Video o Camera, Flash Media Server ofrece una combinación de funciones de flujo de medios y un entorno de desarrollo adecuado para crear y publicar aplicaciones multimedia para un público numeroso. Esta combinación permite que los desarrolladores creen aplicaciones como el vídeo bajo demanda, las difusiones de eventos Web en vivo, las transmisiones de MP3, así como los blogs de vídeo, la mensajería de vídeo y los entornos de chat multimedia. Para obtener más información, consulte la documentación de Flash Media Server en línea en www.adobe.com/go/learn_fms_docs_es.

Temas avanzados para archivos FLV

Los siguientes temas abordan algunos problemas especiales del trabajo con archivos FLV.

Configuración de archivos FLV para alojar en el servidor

Al trabajar con archivos FLV, es posible que sea necesario configurar el servidor para que funcione con el formato de archivo FLV. MIME (Multipurpose Internet Mail Extensions) es una especificación de datos estandarizada que permite enviar archivos que no son ASCII a través de conexiones de Internet. Los navegadores Web y los clientes de correo electrónico se configuran para interpretar numerosos tipos MIME de modo que puedan enviar y recibir vídeo, audio, gráficos y texto con formato. Para cargar archivos FLV de un servidor Web, es posible que se necesite registrar la extensión de archivo y el tipo MIME en el servidor Web, por lo que se debe comprobar la documentación del servidor Web. El tipo MIME de los archivos FLV es `video/x-flv`. La información completa del tipo de archivo FLV es la siguiente:

- Tipo Mime: vídeo/x-flv
- Extensión de archivo: .flv
- Parámetros necesarios: ninguno.
- Parámetros opcionales: ninguno
- Consideraciones de codificación: los archivos FLV son binarios; algunas aplicaciones pueden necesitar establecer el subtipo application/octet-stream.
- Problemas de seguridad: ninguno
- Especificación publicada: www.adobe.com/go/video_file_format_es

Microsoft ha cambiado la forma en la que se controlan los flujos de medios en el servidor Web Servicios de Microsoft Internet Information Server (IIS) 6.0 desde las versiones anteriores. Las versiones anteriores de IIS no necesitan modificaciones para reproducir Flash Video. En IIS 6.0, servidor Web predeterminado que incluye Windows 2003, el servidor necesita un tipo MIME para reconocer que los archivos FLV son flujos de medios.

Cuando los archivos SWF que transmiten archivos FLV externos se sitúan en un servidor Microsoft Windows Server® 2003 y se visualizan en un navegador, el archivo SWF se reproduce correctamente, pero el vídeo FLV no se transmite. Este problema afecta a todos los archivos FLV que se encuentran en Windows Server 2003, incluidos los archivos creados con versiones anteriores de la herramienta de edición de Flash o con Macromedia Flash Video Kit para Dreamweaver MX 2004 de Adobe. Estos archivos funcionan correctamente en otros sistemas operativos.

Para obtener información sobre la configuración de Microsoft Windows 2003 y Microsoft IIS Server 6.0 para transmitir vídeo FLV, consulte www.adobe.com/go/tn_19439_es.

Utilización de archivos FLV locales en Macintosh

Si se intenta reproducir un archivo FLV local desde una unidad que no sea del sistema en un equipo Apple® Macintosh® con una ruta que utilice una barra relativa (/), el vídeo no se reproducirá. Las unidades que no son del sistema incluyen, entre otras, unidades CD-ROM, discos duros con particiones, medios de almacenamiento extraíbles y dispositivos de almacenamiento conectados.

Nota: *el motivo de este error es una limitación del sistema operativo y no de Flash Player o AIR.*

Para que un archivo FLV se reproduzca en una unidad que no sea del sistema en Macintosh, hay que hacer referencia a la misma con una ruta absoluta mediante la notación basada en dos puntos (:) en lugar de la basada en barras (/). La lista siguiente muestra la diferencia de los dos tipos de notación:

- Notación basada en barras**:** myDrive/myFolder/myFLV.flv
- Notación basada en dos puntos**:** (Mac OS®) myDrive:myFolder:myFLV.flv

También se puede crear un archivo de proyector para un CD-ROM que se vaya a utilizar para la reproducción en Macintosh. Para obtener la información más reciente sobre los CD-ROM de Mac OS y los archivos FLV, consulte www.adobe.com/go/3121b301_es.

Ejemplo: Gramola de vídeo

En el ejemplo siguiente se crea un jukebox de vídeo que carga dinámicamente una lista de vídeos que se reproducirán en orden secuencial. Con ello, se crea una aplicación que permite que el usuario examine una serie de tutoriales de vídeo, o tal vez especifica los anuncios que se deben reproducir antes de publicar el vídeo solicitado del usuario. Este ejemplo muestra las siguientes características de ActionScript 3.0:

- Actualizar una cabeza lectora según el progreso de reproducción de un archivo de vídeo
- Detectar y analizar los metadatos de un archivo de vídeo
- Controlar códigos específicos en un objeto NetStream
- Cargar, reproducir, pausar y detener un archivo FLV cargado dinámicamente
- Cambiar el tamaño de un objeto Video en la lista de visualización según los metadatos del objeto NetStream

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Video Jukebox se encuentran en la carpeta Samples/VideoJukebox. La aplicación consta de los siguientes archivos:

Archivo	Descripción
VideoJukebox.fla o VideoJukebox.mxml	Archivo principal de la aplicación de Flex (MXML) o de Flash (FLA).
VideoJukebox.as	La clase que proporciona la funcionalidad principal de la aplicación.
playlist.xml	Archivo que muestra los archivos de vídeo que se cargarán en el jukebox de vídeo.

Carga de un archivo de lista de reproducción de vídeo externo

El archivo playlist.xml externo especifica los vídeos que se cargarán y el orden en que se reproducirán. Para cargar el archivo XML, se debe utilizar un objeto URLLoader y un objeto URLRequest, tal como se muestra en el código siguiente:

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

Este código se coloca en el constructor de la clase VideoJukebox para que el archivo se cargue antes de ejecutar otro código. En cuanto el archivo XML termine de cargarse, se llama al método `xmlCompleteHandler()`, que analiza el archivo externo en un objeto XML, tal como se muestra en el código siguiente:

```
private function xmlCompleteHandler(event:Event):void
{
    playlist = XML(event.target.data);
    videosXML = playlist.video;
    main();
}
```

El objeto XML `playlist` contiene el código XML sin procesar del archivo externo, mientras que videosXML es un objeto XMLList que sólo incluye los nodos de vídeo. En el siguiente fragmento se muestra un archivo playlist.xml de ejemplo:

```
<videos>
    <video url="video/caption_video.flv" />
    <video url="video/cuepoints.flv" />
    <video url="video/water.flv" />
</videos>
```

Por último, el método `xmlCompleteHandler()` llama al método `main()`, que configura las diferentes instancias de componente de la lista de reproducción, así como los objetos NetConnection y NetStream, que se utilizan para cargar los archivos FLV externos.

Creación de la interfaz de usuario

Para crear la interfaz de usuario, es necesario arrastrar cinco instancias de Button a la lista de visualización y asignarles los nombres de instancia siguientes: `playButton`, `pauseButton`, `stopButton`, `backButton` y `forwardButton`.

Para cada una de estas instancias de Button, deberá asignar un controlador para el evento `click`, tal como se muestra en el fragmento siguiente:

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

El método `buttonClickHandler()` utiliza una sentencia switch para determinar la instancia de Button en la que se ha hecho clic, tal como se muestra en el código siguiente:

```
private function buttonClickHandler(event:MouseEvent):void
{
    switch (event.currentTarget)
    {
        case playButton:
            ns.resume();
            break;
        case pauseButton:
            ns.togglePause();
            break;
        case stopButton:
            ns.pause();
            ns.seek(0);
            break;
        case backButton:
            playPreviousVideo();
            break;
        case forwardButton:
            playNextVideo();
            break;
    }
}
```

A continuación, añada una instancia de Slider a la lista de visualización y asígnele el nombre `volumeSlider`. En el código siguiente se establece la propiedad `liveDragging` de la instancia de Slider en `true` y se define un detector de eventos para el evento `change` de dicha instancia:

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

Añada una instancia de ProgressBar a la lista de visualización y asígnele el nombre `positionBar`. Establezca su propiedad `mode` en manual, tal como se muestra en el fragmento siguiente:

```
positionBar.mode = ProgressBarMode.MANUAL;
```

Finalmente, añada una instancia de Label a la lista de visualización y asígnele el nombre `positionLabel`. La instancia de Timer establecerá el valor de la instancia Label.

Detección de los metadatos de un objeto Video

Cuando Flash Player encuentra metadatos para los vídeos cargados, se llama al controlador callback `onMetaData()` en la propiedad `client` del objeto NetStream. En el código siguiente se inicializa un objeto y se configura el controlador callback especificado:

```
client = new Object();
client.onMetaData = metadataHandler;
```

El método `metadataHandler()` copia sus datos en la propiedad meta, definida anteriormente en el código. Esto permite acceder a los metadatos del vídeo actual en cualquier momento y en toda la aplicación. A continuación, se ajusta el tamaño del objeto Video del escenario para que coincida con las dimensiones que devuelven los metadatos. Por último, se desplaza la instancia de ProgressBar positionBar, cuyo tamaño se ajusta según el tamaño del vídeo que se reproduce en ese momento. El código siguiente contiene todo el método `metadataHandler()`:

```
private function metadataHandler(metadataObj:Object):void
{
    meta = metadataObj;
    vid.width = meta.width;
    vid.height = meta.height;
    positionBar.move(vid.x, vid.y + vid.height);
    positionBar.width = vid.width;
}
```

Carga dinámica de un vídeo Flash

Para cargar dinámicamente cada uno de los vídeos Flash, la aplicación utiliza un objeto NetConnection y un objeto NetStream. En el código siguiente se crea un objeto NetConnection y se pasa el valor `null` al método `connect()`. Si se especifica `null`, Flash Player se conecta a un vídeo del servidor local en lugar de conectarse a un servidor, como Flash Media Server.

En el código siguiente se crean las instancias de NetConnection y NetStream, se define un detector de eventos para el evento `netStatus` y se asigna el objeto `client` a la propiedad `client`.

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

Se llama al método `netStatusHandler()` cuando se cambia el estado del vídeo. Esto también se aplica cuando un vídeo inicia o detiene la reproducción, almacena en búfer, o bien si no se encuentra un flujo de vídeo. El código siguiente muestra el evento `netStatusHandler()`:

```
private function netStatusHandler(event:NetStatusEvent):void
{
    try
    {
        switch (event.info.code)
        {
            case "NetStream.Play.Start":
                t.start();
                break;
            case "NetStream.Play.StreamNotFound":
            case "NetStream.Play.Stop":
                t.stop();
                playNextVideo();
                break;
        }
    }
    catch (error:TypeError)
    {
        // Ignore any errors.
    }
}
```

El código anterior evalúa la propiedad de código del objeto info y filtra si el código es "NetStream.Play.Start", "NetStream.Play.StreamNotFound" o "NetStream.Play.Stop". Los otros códigos se omitirán. Si el objeto NetStream inicia el código, se inicia la instancia de Timer que actualiza la cabeza lectora. Si el objeto NetStream no se puede encontrar o se detiene, la instancia de Timer se detiene y la aplicación intenta reproducir el siguiente vídeo de la lista de reproducción.

Cada vez que se ejecuta Timer, la instancia de ProgressBar `positionBar` actualiza su posición actual llamando al método `setProgress()` de la clase ProgressBar, y la instancia de Label `positionLabel` se actualiza con el tiempo transcurrido y el tiempo total del vídeo actual.

```
private function timerHandler(event:TimerEvent):void
{
    try
    {
        positionBar.setProgress(ns.time, meta.duration);
        positionLabel.text = ns.time.toFixed(1) + " of " meta.duration.toFixed(1) + " seconds";
    }
    catch (error:Error)
    {
        // Ignore this error.
    }
}
```

Control del volumen de vídeo

Se puede controlar el volumen del vídeo cargado dinámicamente estableciendo la propiedad `soundTransform` del objeto NetStream. La aplicación de jukebox de vídeo permite modificar el nivel de volumen cambiando el valor `volumeSlider` de la instancia Slider. En el código siguiente se muestra cómo se puede cambiar el nivel de volumen asignando el valor del componente Slider a un objeto SoundTransform, que se establece en la propiedad `soundTransform` del objeto NetStream:

```
private function volumeChangeHandler(event:SliderEvent):void
{
    volumeTransform.volume = event.value;
    ns.soundTransform = volumeTransform;
}
```

Control de la reproducción de vídeo

El resto de la aplicación controla la reproducción de vídeo cuando éste alcanza el final del flujo de vídeo, o bien cuando el usuario salta al vídeo anterior o siguiente.

En el método siguiente se recupera el URL de vídeo del objeto XMLList para el índice seleccionado en ese momento:

```
private function getVideo():String
{
    return videosXML[idx].@url;
}
```

El método `playVideo()` llama al método `play()` del objeto NetStream para cargar el vídeo seleccionado actualmente:

```
private function playVideo():void
{
    var url:String = getVideo();
    ns.play(url);
}
```

El método `playPreviousVideo()` reduce el índice de vídeo actual, llama al método `playVideo()` para cargar el nuevo archivo de vídeo y establece la instancia de ProgressBar en visible:

```
private function playPreviousVideo():void
{
    if (idx > 0)
    {
        idx--;
        playVideo();
        positionBar.visible = true;
    }
}
```

El método final, `playNextVideo()`, incrementa el índice de vídeo y llama al método `playVideo()`. Si el vídeo actual es el último de la lista de reproducción, se llama al método `clear()` del objeto Video y la propiedad `visible` de la instancia de ProgressBar se establece en `false`:

```
private function playNextVideo():void
{
    if (idx < (videosXML.length() - 1))
    {
        idx++;
        playVideo();
        positionBar.visible = true;
    }
    else
    {
        idx++;
        vid.clear();
        positionBar.visible = false;
    }
}
```

Capítulo 25: Trabajo con sonido

ActionScript se ha diseñado para crear aplicaciones interactivas y envolventes. Un elemento a menudo olvidado de las aplicaciones envolventes es el sonido. Se pueden añadir efectos de sonido a un videojuego, comentarios de audio a una interfaz de usuario o incluso crear un programa que analice archivos MP3 cargados por Internet, con el sonido como un componente importante de la aplicación.

En este capítulo se describe la manera de cargar archivos de audio externos y trabajar con audio incorporado en un archivo SWF. Asimismo, se muestra cómo controlar el audio, crear representaciones visuales de la información de sonido y capturar el sonido desde un micrófono del usuario.

Fundamentos de la utilización de sonido

Introducción a la utilización de sonido

Los equipos pueden capturar y codificar audio digital (representación mediante ordenador de información de sonido) y almacenarlo y recuperarlo para su reproducción con altavoces. El sonido se puede reproducir utilizando Adobe® Flash® Player o Adobe® AIR™ y ActionScript.

Los datos de sonido convertidos a formato digital tienen varias características, como el volumen del sonido y si éste es estéreo o mono. Al reproducir un sonido en ActionScript, también se pueden ajustar estas características (por ejemplo, aumentar su volumen o hacer que parezca proceder de una dirección determinada).

Para poder controlar un sonido en ActionScript es necesario tener la información de sonido cargada en Flash Player o AIR. Hay cuatro cinco de obtener datos de audio en Flash Player o AIR y trabajar con ellos mediante ActionScript. Se puede cargar un archivo de sonido externo, como un archivo MP3, en el archivo SWF; la información de sonido se puede incorporar al archivo SWF directamente cuando se crea. Se puede obtener una entrada de audio con un micrófono conectado al ordenador del usuario, así como acceder a los datos que se transmiten desde un servidor; se puede trabajar con datos de sonido que se generan dinámicamente.

Al cargar datos de sonido desde un archivo de sonido externo, se puede iniciar la reproducción del principio del archivo de sonido mientras se cargan los restantes datos.

Aunque hay varios formatos de archivo de sonido que se utilizan para codificar audio digital, ActionScript 3.0, Flash Player y AIR admiten archivos de sonido almacenados en formato MP3. No pueden cargar ni reproducir directamente archivos de sonido con otros formatos, como WAV o AIFF.

Al trabajar con sonido en ActionScript, es probable que se utilicen varias clases del paquete flash.media. La clase Sound se utiliza para acceder a la información de audio mediante la carga de un archivo de sonido o la asignación de una función a un evento que muestrea los datos de sonido y posteriormente se inicia la reproducción. Una vez iniciada la reproducción de un sonido, Flash Player y AIR proporcionan acceso a un objeto SoundChannel. Puesto que un archivo de audio que se ha cargado puede ser uno de varios sonidos que se reproducen en el ordenador de un usuario, cada sonido que se reproduce utiliza su propio objeto SoundChannel; la salida combinada de todos los objetos SoundChannel mezclados es lo que se reproduce realmente a través de los altavoces del ordenador. La instancia de SoundChannel se utiliza para controlar las propiedades del sonido y detener su reproducción. Por último, si se desea controlar el audio combinado, la clase SoundMixer permite controlar la salida mezclada.

Asimismo, se pueden utilizar otras muchas clases para realizar tareas más específicas cuando se trabaja con sonido en ActionScript; para más información sobre todas las clases relacionadas con el sonido, consulte"Aspectos básicos de la arquitectura de sonido" en la página 581.

Tareas comunes relacionadas con el sonido

En este capítulo se describen las siguientes tareas relacionadas con el sonido que se realizan frecuentemente:

- Cargar archivos MP3 externos y hacer un seguimiento de su progreso de carga
- Reproducir, pausar, reanudar y detener sonidos
- Reproducir flujos de sonido mientras se cargan
- Manipulación de desplazamiento y volumen de sonido
- Recuperar metadatos ID3 de un archivo MP3
- Utilizar datos de onda de sonido sin formato
- Sonido de generación dinámica
- Capturar y reproducir entradas de sonido del micrófono de un usuario

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Amplitud: distancia de un punto de la forma de onda del sonido desde la línea cero o de equilibrio.
- Velocidad de bits: cantidad de datos que se codifican o se transmiten en cada segundo de un archivo de sonido. En los archivos MP3, la velocidad suele expresarse en miles de bits por segundo (kbps). Una velocidad superior suele implicar una onda de sonido de mayor calidad.
- Almacenamiento en búfer: recepción y almacenamiento de los datos de sonido antes de que se reproduzcan.
- mp3: MPEG-1 Audio Layer 3, o MP 3, es un formato de compresión de sonido conocido.
- Desplazamiento lateral: posición de una señal de audio entre los canales izquierdo y derecho de un campo de sonido estéreo.
- Pico: punto más alto de una forma de onda.
- Velocidad de muestreo: define el número de muestras por segundo que se toman de una señal de audio analógica para crear una señal digital. La velocidad de muestra del audio de CD estándar es de 44,1 kHz o 44.100 muestras por segundo.
- Transmisión: proceso de reproducir el principio de un archivo de sonido o de vídeo mientras se descargan los datos restantes desde un servidor.
- Volumen: intensidad de un sonido.
- Forma de onda: forma de un gráfico de las distintas amplitudes de una señal de sonido a lo largo del tiempo.

Ejecución de los ejemplos del capítulo

A medida que progrese en el estudio de este capítulo, podría desear probar algunos de los listados de código de ejemplo. Como en este capítulo se describe la utilización de sonido en ActionScript, muchos de los ejemplos realizan tareas que requieren trabajar con un archivo de sonido: reproducirlo, detener la reproducción o ajustar el sonido de alguna manera. Para probar los ejemplos de este capítulo:

1 Cree un nuevo documento de Flash y guárdelo en su equipo.

2 En la línea de tiempo, seleccione el primer fotograma clave y abra el panel Acciones.

3 Copie el ejemplo de código en el panel Script.

4 Si el código requiere cargar un archivo de sonido externo, tendrá una línea de código similar a la siguiente:

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
s.play();
```

donde "click.mp3" es el nombre del archivo de sonido que se va cargar. Para probar estos ejemplos necesita un archivo MP3. Debe colocar el archivo MP3 en la misma carpeta que el documento de Flash. Después debe modificar el código para que utilice el nombre del archivo MP3 en lugar del nombre especificado en el listado de código (por ejemplo, en el código anterior, debe cambiar "click.mp3" por el nombre de su archivo MP3).

5 En el menú principal, elija Control > Probar película para crear el archivo SWF y mostrar una vista previa de la salida del ejemplo (además de escucharla).

Además de reproducir de audio, parte de los ejemplos utilizan la función `trace()` para mostrar valores; cuando pruebe los ejemplos verá los resultados en el panel Salida. Algunos ejemplos también dibujan contenido en la pantalla; para estos ejemplos también verá el contenido en la ventana de Flash Player o AIR.

Para obtener más información sobre la prueba de listados de código de ejemplo en este manual, consulte “Prueba de los listados de código de ejemplo del capítulo” en la página 36.

Aspectos básicos de la arquitectura de sonido

Las aplicaciones pueden cargar datos de sonido de cinco orígenes principales:

- Archivos de sonido externos cargados en tiempo de ejecución
- Recursos de sonido incorporados en el archivo SWF de la aplicación
- Datos de sonido de un micrófono conectado al sistema del usuario
- Datos de sonido transmitidos desde un servidor multimedia remoto, como Flash Media Server
- Datos de sonido generados dinámicamente mediante el uso del controlador de eventos `sampleData`.

Los datos de sonido se pueden cargar completamente antes de su reproducción, o bien se pueden transmitirse, es decir, se pueden reproducir mientras se están cargando.

Las clases de sonido de ActionScript 3.0 admiten archivos de sonido que se almacenan en formato mp3. No pueden cargar ni reproducir directamente archivos de sonido con otros formatos, como WAV o AIFF. No obstante, cuando se comienza con Flash Player 9.0.115.0, los archivos de audio AAC se pueden cargar y reproducir utilizando la clase NetStream. Se trata de la misma técnica que se utiliza para cargar y reproducir contenido de vídeo. Para obtener más información sobre esta técnica, consulte “Trabajo con vídeo” en la página 538.

Adobe Flash CS4 Professional permite importar archivos de sonido WAV o AIFF e incorporarlos en los archivos SWF de la aplicación con formato MP3. La herramienta de edición de Flash también permite comprimir archivos de sonido incorporados para reducir su tamaño de archivo, aunque esta reducción de tamaño se compensa con una mejor calidad de sonido. Para más información, consulte "Importación de sonidos" en *Utilización de Flash*.

La arquitectura de sonido de ActionScript 3.0 utiliza las siguientes clases del paquete flash.media.

Clase	Descripción
flash.media.Sound	La clase Sound controla la carga del sonido, administra las propiedades de sonido básicas e inicia la reproducción de sonido.
flash.media.SoundChannel	Cuando una aplicación reproduce un objeto Sound, se crea un nuevo objeto SoundChannel para controlar la reproducción. El objeto SoundChannel controla el volumen de los canales de reproducción izquierdo y derecho del sonido. Cada sonido que se reproduce tiene su propio objeto SoundChannel.
flash.media.SoundLoaderContext	La clase SoundLoaderContext especifica cuántos segundos de búfer se utilizarán al cargar un sonido, y si Flash Player o AIR buscan un archivo de política en el servidor al cargar un archivo. Un objeto SoundLoaderContext se utiliza como parámetro del método `Sound.load()`.
flash.media.SoundMixer	La clase SoundMixer controla la reproducción y las propiedades de seguridad que pertenecen a todos los sonidos de una aplicación. De hecho, se mezclan varios canales de sonido mediante un objeto SoundMixer común, de manera que los valores de propiedad de dicho objeto afectarán a todos los objetos SoundChannel que se reproducen actualmente.
flash.media.SoundTransform	La clase SoundTransform contiene valores que controlan el desplazamiento y el volumen de sonido. Un objeto SoundTransform puede aplicarse a un objeto SoundChannel, al objeto SoundMixer global o a un objeto Microphone, entre otros.
flash.media.ID3Info	Un objeto ID3Info contiene propiedades que representan información de metadatos ID3, almacenada a menudo en archivos de sonido MP3.
flash.media.Microphone	La clase Microphone representa un micrófono u otro dispositivo de entrada de sonido conectado al ordenador del usuario. La entrada de audio de un micrófono puede enrutarse a altavoces locales o enviarse a un servidor remoto. El objeto Microphone controla la ganancia, la velocidad de muestra y otras características de su propio flujo de sonido.

Cada sonido que se carga y se reproduce necesita su propia instancia de las clases Sound y SoundChannel. La clase SoundMixer global mezcla la salida de varias instancias de SoundChannel durante la reproducción.

Las clases Sound, SoundChannel y SoundMixer no se utilizan para datos de sonido obtenidos de un micrófono o de un servidor de flujo de medios como Flash Media Server.

Carga de archivos de sonido externos

Cada instancia de la clase Sound existe para cargar y activar la reproducción de un recurso de sonido específico. Una aplicación no puede reutilizar un objeto Sound para cargar más de un sonido. Para cargar un nuevo recurso de sonido, debe crear otro objeto Sound.

Si se carga un archivo de sonido pequeño, como un sonido de clic que se asociará a un botón, la aplicación puede crear un nuevo objeto Sound y cargar automáticamente el archivo de sonido, tal como se muestra a continuación:

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

El constructor `Sound()` acepta un objeto URLRequest como primer parámetro. Cuando se proporciona un valor del parámetro URLRequest, el nuevo objeto Sound empieza a cargar automáticamente el recurso de sonido especificado.

En todos los casos, excepto en los más sencillos, la aplicación debe prestar atención al progreso de carga del sonido y detectar los posibles errores. Por ejemplo, si el sonido del clic tiene un tamaño bastante grande, es posible que no esté completamente cargado cuando el usuario haga clic en el botón que activa dicho sonido. Si se intenta reproducir un sonido no cargado, puede producirse un error en tiempo de ejecución. Resulta más seguro esperar a que se cargue totalmente el sonido antes de permitir que los usuarios realicen acciones que puedan iniciar la reproducción de sonidos.

El objeto Sound distribuye varios eventos distintos durante el proceso de carga del sonido. La aplicación puede detectar estos eventos para hacer un seguimiento del progreso de carga y asegurarse de que el sonido se carga por completo antes de reproducirse. En la tabla siguiente se enumeran los eventos que puede distribuir un objeto Sound.

Evento	Descripción
open (`Event.OPEN`)	Se distribuye antes de que se inicie la operación de carga del sonido.
progress (`ProgressEvent.PROGRESS`)	Se distribuye periódicamente durante el proceso de carga del sonido cuando se reciben datos del archivo o del flujo.
id3 (`Event.ID3`)	Se distribuye cuando hay datos ID3 disponibles para un sonido MP3.
complete (`Event.COMPLETE`)	Se distribuye cuando se han cargado todos los datos del recurso de sonido.
ioError (`IOErrorEvent.IO_ERROR`)	Se distribuye cuando no se puede encontrar un archivo de sonido, o bien cuando el proceso de carga se interrumpe antes de que se puedan recibir todos los datos de sonido.

El código siguiente ilustra la manera de reproducir un sonido una vez cargado:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

En primer lugar, el código de ejemplo crea un nuevo objeto Sound sin asignarle un valor inicial para el parámetro URLRequest. A continuación, detecta el evento `Event.COMPLETE` del objeto Sound, que provoca la ejecución del método `onSoundLoaded()` cuando se cargan todos los datos de sonido. Después, llama al método `Sound.load()` con un nuevo valor URLRequest para el archivo de sonido.

El método `onSoundLoaded()` se ejecuta cuando se completa la carga de sonido. La propiedad `target` del objeto Event es una referencia al objeto Sound. La llamada al método `play()` del objeto Sound inicia la reproducción de sonido.

Supervisión del proceso de carga del sonido

Los archivos de sonido pueden tener un tamaño muy grande y tardar mucho tiempo en cargarse. Aunque Flash Player y AIR permiten que la aplicación reproduzca sonidos incluso antes de que se carguen completamente, es posible que se desee proporcionar una indicación al usuario de la cantidad de datos de sonido que se han cargado, así como de la cantidad de sonido que ya se ha reproducido.

La clase Sound distribuye dos eventos que hacen relativamente fácil la visualización del proceso de carga de un sonido: `ProgressEvent.PROGRESS` y `Event.COMPLETE`. En el ejemplo siguiente se muestra cómo utilizar estos eventos para mostrar la información de progreso del sonido que se carga:

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
    var loadedPct:uint = Math.round(100 * (event.bytesLoaded / event.bytesTotal));
    trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
function onIOError(event:IOErrorEvent)
{
    trace("The sound could not be loaded: " + event.text);
}
```

En este código se crea primero un objeto Sound y luego se añaden detectores a dicho objeto para los eventos `ProgressEvent.PROGRESS` y `Event.COMPLETE`. Una vez que el método `Sound.load()` se ha llamado y que el archivo de sonido recibe los primeros datos, se produce un evento `ProgressEvent.PROGRESS`, que activa el método `onSoundLoadProgress()`.

El porcentaje de datos de sonido cargados equivale al valor de la propiedad `bytesLoaded` del objeto ProgressEvent dividido entre el valor de la propiedad `bytesTotal`. Las mismas propiedades `bytesLoaded` y `bytesTotal` también están disponibles en el objeto Sound. El ejemplo anterior muestra mensajes relativos al progreso de carga del sonido, aunque se pueden utilizar fácilmente los valores `bytesLoaded` y `bytesTotal` para actualizar los componentes de barra de progreso, como los que se incluyen en la arquitectura Adobe Flex 3 o en la herramienta de edición Flash.

En este ejemplo también se muestra la forma en que una aplicación puede reconocer un error y responder a él durante la carga de archivos de sonido. Por ejemplo, si no se puede encontrar un archivo de sonido con un nombre determinado, el objeto Sound distribuye un evento `Event.IO_ERROR`. En el código anterior, se ejecuta el método `onIOError()`, que muestra un breve mensaje de error cuando se produce un error.

Trabajo con sonidos incorporados

La utilización de sonidos incorporados, en lugar de cargar sonidos de un archivo externo, resulta útil para sonidos de pequeño tamaño que se emplean como indicadores en la interfaz de usuario de la aplicación (por ejemplo, los sonidos que se reproducen cuando se hace clic en los botones).

Si se incorpora un archivo de sonido en la aplicación, el tamaño del archivo SWF resultante aumenta con el tamaño del archivo de sonido. Es decir, la incorporación de archivos de sonido grandes en la aplicación puede hacer que el tamaño del archivo SWF sea excesivo.

El método preciso de incorporar un archivo de sonido en el archivo SWF de la aplicación varía de un entorno de desarrollo a otro.

Utilización de un archivo de sonido incorporado en Flash

La herramienta de edición de Flash permite importar sonidos en varios formatos de sonido y almacenarlos como símbolos en la Biblioteca. Después se pueden asignar a fotogramas en la línea de tiempo o a los fotogramas de un estado de botón, utilizarlos con Comportamientos o utilizarlos directamente en código ActionScript. En esta sección se describe la manera de utilizar sonidos incorporados en código ActionScript con la herramienta de edición de Flash. Para más información sobre las otras maneras de utilizar sonidos incorporados en Flash, consulte "Importación de sonidos" en *Utilización de Flash*.

Para incorporar un archivo de sonido utilizando la herramienta de edición de Flash:

1. Seleccione Archivo > Importar > Importar a biblioteca y, a continuación, elija un archivo de sonido para importarlo.
2. Haga clic con el botón derecho en el nombre del archivo importado en el panel Biblioteca y seleccione Propiedades. Active la casilla de verificación Exportar para ActionScript.
3. En el campo Clase, escriba el nombre que debe utilizarse al hacer referencia a este sonido incorporado en ActionScript. De manera predeterminada, utilizará el nombre del archivo de sonido de este campo. Si el nombre de archivo incluye un punto, como en "DrumSound.mp3", debe cambiarlo por otro como "DrumSound"; ActionScript no permite utilizar un punto en un nombre de clase. El campo Clase base debe mostrar flash.media.Sound.
4. Haga clic en Aceptar. Es posible que vea un cuadro de mensaje que indique que no se encontró una definición para esta clase en la ruta de clases. Haga clic en Aceptar y continúe. Si se introduce una clase cuyo nombre no coincide con el de ninguna de las clases de la ruta de clases de la aplicación, se genera automáticamente una nueva clase que hereda de la clase flash.media.Sound.
5. Para utilizar el sonido incorporado hay que hacer referencia al nombre de clase del sonido en ActionScript. Por ejemplo, el código siguiente empieza creando una nueva instancia de la clase DrumSound generada automáticamente:

   ```
   var drum:DrumSound = new DrumSound();
   var channel:SoundChannel = drum.play();
   ```

   DrumSound es una subclase de la clase flash.media.Sound, por lo que hereda los métodos y las propiedades de la clase Sound, incluido el método `play()` antes mostrado.

Trabajo con archivos de flujo de sonido

Si se reproduce un archivo de sonido o de vídeo mientras se cargan sus datos, se dice que se está *transmitiendo* el archivo. Los archivos de sonido externos que se cargan desde un servidor remoto suelen transmitirse; de este modo, no es necesario que el usuario espere hasta que se carguen todos los datos para poder escuchar el sonido.

La propiedad `SoundMixer.bufferTime` representa el número de milisegundos de datos de sonido que deben recopilar Flash Player o AIR antes de permitir la reproducción de sonido. Es decir, si la propiedad `bufferTime` se establece en 5000, Flash Player o AIR cargan al menos 5000 milisegundos (el equivalente en datos) del archivo de sonido antes de empezar a reproducir el sonido. El valor predeterminado de `SoundMixer.bufferTime` es 1000.

La aplicación puede sustituir el valor global de `SoundMixer.bufferTime` para un sonido determinado especificando explícitamente un nuevo valor de `bufferTime` al cargar el sonido. Para sustituir el tiempo de búfer predeterminado, primero hay que crear una nueva instancia de la clase SoundLoaderContext, establecer el valor de su propiedad `bufferTime` y pasarla como un parámetro al método `Sound.load()`, tal como se muestra a continuación:

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

Mientras sigue la reproducción, Flash Player y AIR intentan mantener el tamaño del búfer de sonido en un valor igual o mayor. Si los datos de sonido se cargan más rápido que la velocidad de reproducción, ésta continuará sin interrupciones. Sin embargo, si la velocidad de carga de los datos disminuye debido a limitaciones de red, la cabeza lectora podría alcanzar el final del búfer de sonido. Si esto sucede, se suspende la reproducción, aunque se reanudará automáticamente en cuanto se hayan cargado más datos de sonido.

Para determinar si se ha suspendido la reproducción porque Flash o AIR están esperando a que se carguen más datos, se debe utilizar la propiedad `Sound.isBuffering`.

Trabajo con sonido generado dinámicamente

Nota: *la posibilidad de generar audio dinámicamente está presente a partir de Flash Player 10 y Adobe AIR 1.5.*

En lugar de cargar o transmitir un sonido existente, se pueden generar dinámicamente los datos de audio. Puede generar datos de audio cuando se asigna un detector de eventos para el evento `sampleData` de un objeto Sound. (El evento `sampleData` se define en la clase SampleDataEvent en el paquete flash.events.) En este entorno, el objeto Sound no carga datos de sonido desde un archivo. Actúa como socket para datos de sonido transmitidos en flujo mediante la función asignada al evento.

Cuando se añade un detector de eventos `sampleData` a un objeto Sound, el objeto solicita datos periódicamente para agregarlos al búfer de sonido. Este búfer contiene datos para el objeto Sound que se va a reproducir. Cuando se llama al método `play()` del objeto Sound, se distribuye el evento `sampleData` al solicitar nuevos datos de sonido. (El valor es true sólo cuando el objeto Sound no ha cargado datos mp3 desde un archivo.)

El objeto SampleDataEvent incluye una propiedad `data`. En el detector de eventos, los objetos ByteArray se escriben en este objeto`data`. Los conjuntos de bytes en los que se escribe este objeto se añaden a los datos de sonido almacenados en búfer que reproduce el objeto Sound. El conjunto de bytes en el búfer es un flujo de valores de coma flotante de -1 a 1. Cada valor de coma flotante representa al amplitud de un canal (izquierdo o derecho) de una muestra de sonido. El sonido se muestrea a 44.100 muestras por segundo. Cada muestra contiene un canal izquierdo y derecho, intercalada como datos de coma flotante en el conjunto de bytes.

En la función de controlador, se utiliza el método `ByteArray.writeFloat()` para escribir en la propiedad `data` del evento `sampleData`. Por ejemplo, el siguiente código genera una onda de seno:

```
var mySound:Sound = new Sound();
mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, sineWaveGenerator);
mySound.play();
function sineWaveGenerator(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192; i++)
    {
        var n:Number = Math.sin((i + event.position) / Math.PI / 4);
        event.data.writeFloat(n);
        event.data.writeFloat(n);
    }
}
```

Cuando se llama a `Sound.play()`, la aplicación inicia la llamada al controlador de eventos, solicitando los datos de la muestra de sonido. La aplicación continúa enviando eventos a medida que se reproduce el sonido hasta que se detiene el suministro o se llama a `SoundChannel.stop()`.

La latencia del evento varía en función de la plataforma y puede cambiar en futuras versiones de Flash Player y AIR. No dependa de una latencia concreta: calcúlela. Para calcular la latencia, utilice la siguiente fórmula:

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

Proporcione de 2048 hasta 8192 muestras a la propiedad `data` del objeto SampleDataEvent (para cada llamada al controlador de eventos). Para obtener el mejor rendimiento, proporcione tantas muestras como sea posible (hasta 8192). Cuanto menor sea el número de muestras, más probable será la aparición de interferencias en la reproducción. Este comportamiento puede variar en función de la plataforma y según las situaciones; por ejemplo, al cambiar de tamaño la ventana del navegador. El código que funciona en una plataforma cuando sólo se proporcionan 2048 muestras puede que no funcione tan bien cuando se ejecuta en una plataforma distinta. Si necesita la menor latencia posible, considere la posibilidad de que sea el usuario quien seleccione la cantidad de datos.

Si se proporcionan menos de 2048 muestras (por llamada al detector de eventos `sampleData`), la aplicación se detiene tras reproducir las muestras restantes. Posteriormente distribuye un evento SoundComplete.

Modificación de sonido desde datos mp3

El método `Sound.extract()` se utiliza para extraer datos de un objeto Sound. Puede utilizar (y modificar) los datos para escribir en el flujo dinámico de otro objeto Sound para la reproducción. Por ejemplo, el siguiente código utiliza los bytes de archivo MP3 cargado y los transmite mediante una función de filtro, `upOctave()`:

```
var mySound:Sound = new Sound();
var sourceSnd:Sound = new Sound();
var urlReq:URLRequest = new URLRequest("test.mp3");
sourceSnd.load(urlReq);
sourceSnd.addEventListener(Event.COMPLETE, loaded);
function loaded(event:Event):void
{
    mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, processSound);
    mySound.play();
}
function processSound(event:SampleDataEvent):void
{
        var bytes:ByteArray = new ByteArray();
        sourceSnd.extract(bytes, 8192);
        event.data.writeBytes(upOctave(bytes));
}
function upOctave(bytes:ByteArray):ByteArray
{
    var returnBytes:ByteArray = new ByteArray();
    bytes.position = 0;
    while(bytes.bytesAvailable > 0)
    {
        returnBytes.writeFloat(bytes.readFloat());
        returnBytes.writeFloat(bytes.readFloat());
        if (bytes.bytesAvailable > 0)
        {
            bytes.position += 8;
        }
    }
    return returnBytes;
}
```

Limitaciones en los sonidos generados

Cuando se utiliza un detector de eventos `sampleData` con un objeto Sound, los únicos métodos de Sound que permanecen activados son`Sound.extract()` y `Sound.play()`. Llamar a otros métodos o propiedades genera una excepción. Todos los métodos y propiedades del objeto SoundChannel siguen activados.

Reproducción de sonidos

Para reproducir un sonido cargado sólo hay que llamar al método `Sound.play()` de un objeto Sound:

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

Al reproducir sonidos con ActionScript 3.0, se pueden realizar las operaciones siguientes:

- Reproducir un sonido desde una posición de inicio específica
- Pausar un sonido y reanudar la reproducción desde la misma posición posteriormente
- Saber exactamente cuándo termina de reproducirse un sonido
- Hacer un seguimiento del progreso de la reproducción de un sonido
- Cambiar el volumen o el desplazamiento cuando se reproduce un sonido

Para realizar estas operaciones durante la reproducción deben utilizarse las clases SoundChannel, SoundMixer y SoundTransform.

La clase SoundChannel controla la reproducción de un solo sonido. La propiedad `SoundChannel.position` puede considerarse como una cabeza lectora, que indica el punto actual en los datos de sonido que se están reproduciendo.

Cuando una aplicación llama al método `Sound.play()`, se crea una nueva instancia de la clase SoundChannel para controlar la reproducción.

La aplicación puede reproducir un sonido desde una posición de inicio específica, pasando dicha posición en milisegundos como el parámetro `startTime` del método `Sound.play()`. También puede especificar un número fijo de veces que se repetirá el sonido en una sucesión rápida pasando un valor numérico en el parámetro `loops` del método `Sound.play()`.

Cuando se llama al método `Sound.play()` con los parámetros `startTime` y `loops`, el sonido se reproduce de forma repetida desde el mismo punto de inicio cada vez, tal como se muestra en el código siguiente:

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

En este ejemplo, el sonido se reproduce desde un punto un segundo después del inicio del sonido, tres veces seguidas.

Pausa y reanudación de un sonido

Si la aplicación reproduce sonidos largos, como canciones o emisiones podcast, es recomendable dejar que los usuarios pausen y reanuden la reproducción de dichos sonidos. Durante la reproducción en ActionScript un sonido no se puede pausar; sólo se puede detener. Sin embargo, un sonido puede reproducirse desde cualquier punto. Se puede registrar la posición del sonido en el momento en que se detuvo y volver a reproducir el sonido desde dicha posición posteriormente.

Por ejemplo, supongamos que el código carga y reproduce un archivo de sonido como este:

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

Mientras se reproduce el sonido, la propiedad `SoundChannel.position` indica el punto del archivo de sonido que se está reproduciendo en ese momento. La aplicación puede almacenar el valor de posición antes de detener la reproducción del sonido, como se indica a continuación:

```
var pausePosition:int = channel.position;
channel.stop();
```

Para reanudar la reproducción del sonido desde el mismo punto en que se detuvo, hay que pasar el valor de la posición almacenado anteriormente.

```
channel = snd.play(pausePosition);
```

Control de la reproducción

Es posible que la aplicación desee saber cuándo se deja de reproducir un sonido para poder iniciar la reproducción de otro o para limpiar algunos recursos utilizados durante la reproducción anterior. La clase SoundChannel distribuye un evento `Event.SOUND_COMPLETE` cuando finaliza la reproducción de un sonido. La aplicación puede detectar este evento y adoptar las medidas oportunas como se muestra a continuación:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("smallSound.mp3");
snd.load(req);

var channel:SoundChannel = snd.play();
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
}
```

La clase SoundChannel no distribuye eventos de progreso durante la reproducción. Para informar sobre el progreso de la reproducción, la aplicación puede configurar su propio mecanismo de sincronización y hacer un seguimiento de la posición de la cabeza lectora del sonido.

Para calcular el porcentaje de sonido que se ha reproducido, se puede dividir el valor de la propiedad `SoundChannel.position` entre la duración de los datos del sonido que se está reproduciendo:

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

Sin embargo, este código sólo indica porcentajes de reproducción precisos si los datos de sonido se han cargado completamente antes del inicio de la reproducción. La propiedad `Sound.length` muestra el tamaño de los datos de sonido que se cargan actualmente, no el tamaño definitivo de todo el archivo de sonido. Para hacer un seguimiento del progreso de reproducción de un flujo de sonido, la aplicación debe estimar el tamaño final de todo el archivo de sonido y utilizar dicho valor en sus cálculos. Se puede estimar la duración final de los datos de sonido mediante las propiedades `bytesLoaded` y `bytesTotal` del objeto Sound, de la manera siguiente:

```
var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

El código siguiente carga un archivo de sonido más grande y utiliza el evento `Event.ENTER_FRAME` como mecanismo de sincronización para mostrar el progreso de la reproducción. Notifica periódicamente el porcentaje de reproducción, que se calcula como el valor de la posición actual dividido entre la duración total de los datos de sonido:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
    URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

function onEnterFrame(event:Event):void
{
    var estimatedLength:int =
        Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
    var playbackPercent:uint =
        Math.round(100 * (channel.position / estimatedLength));
    trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Una vez iniciada la carga de los datos de sonido, este código llama al método `snd.play()` y almacena el objeto SoundChannel resultante en la variable `channel`. A continuación, añade un detector de eventos a la aplicación principal para el evento `Event.ENTER_FRAME` y otro detector de eventos al objeto SoundChannel para el evento `Event.SOUND_COMPLETE` que se produce cuando finaliza la reproducción.

Cada vez que la aplicación alcanza un nuevo fotograma en su animación, se llama al método `onEnterFrame()`. El método `onEnterFrame()` estima la duración total del archivo de sonido según la cantidad de datos que ya se han cargado, y luego calcula y muestra el porcentaje de reproducción actual.

Cuando se ha reproducido todo el sonido, se ejecuta el método `onPlaybackComplete()` y se elimina el detector de eventos para el evento `Event.ENTER_FRAME`, de manera que no intente mostrar actualizaciones de progreso tras el fin de la reproducción.

El evento `Event.ENTER_FRAME` puede distribuirse muchas veces por segundo. En algunos casos no será necesario mostrar el progreso de la reproducción con tanta frecuencia. De ser así, la aplicación puede configurar su propio mecanismo de sincronización con la clase flash.util.Timer; véase "Trabajo con fechas y horas" en la página 135.

Detener flujos de sonido

Existe una anomalía en el proceso de reproducción para los sonidos que se están transmitiendo, es decir, para los sonidos que todavía se están cargando mientras se reproducen. Cuando la aplicación llama al método `SoundChannel.stop()` en una instancia de SoundChannel que reproduce un flujo de sonido, la reproducción se detiene en un fotograma y en el siguiente se reinicia desde el principio del sonido. Esto sucede porque el proceso de carga del sonido todavía está en curso. Para detener la carga y la reproducción de un flujo de sonido, llame al método `Sound.close()`.

Consideraciones de seguridad al cargar y reproducir sonidos

La capacidad de la aplicación para acceder a datos de sonido puede limitarse según el modelo de seguridad de Flash Player o AIR. Cada sonido está sujeto a las restricciones de dos entornos limitados de seguridad diferentes, el del contenido en sí (el "entorno limitado de contenido") y el de la aplicación o el objeto que carga y reproduce el sonido (el "entorno limitado de propietario"). Para el contenido de la aplicación AIR en el entorno limitado de seguridad de la aplicación, todos los sonidos, incluidos los que se cargan desde otros dominios, están accesibles para el contenido en el entorno limitado de seguridad de la aplicación. Sin embargo, el contenido de otros entornos limitados de seguridad sigue las mismas reglas que el contenido ejecutado en Flash Player. Para más información sobre el modelo de seguridad de Flash Player en general y la definición de los entornos limitados, consulte “Seguridad de Flash Player” en la página 714.

El entorno limitado de contenido controla si se pueden extraer datos detallados del sonido con la propiedad `id3` o el método `SoundMixer.computeSpectrum()`. No limita la carga o la reproducción del sonido en sí.

El dominio de origen del archivo de sonido define las limitaciones de seguridad del entorno limitado de contenido. Normalmente, si un archivo de sonido se encuentra en el mismo dominio o la misma carpeta que el archivo SWF de la aplicación o el objeto que lo carga, éstos tendrán acceso total al archivo de sonido. Si el sonido procede de un dominio diferente al de la aplicación, aún es posible extraerlo al entorno limitado de contenido mediante un archivo de política.

La aplicación puede pasar un objeto SoundLoaderContext con una propiedad `checkPolicyFile` como parámetro al método `Sound.load()`. Si se establece la propiedad `checkPolicyFile` en `true`, se indica a Flash Player o AIR que busque un archivo de política en el servidor desde el que se carga el sonido. Si ya existe un archivo de política, y proporciona acceso al dominio del archivo SWF que se carga, el archivo SWF puede cargar el archivo de sonido, acceder a la propiedad `id3` del objeto Sound y llamar al método `SoundMixer.computeSpectrum()` para sonidos cargados.

El entorno limitado de propietario controla la reproducción local de los sonidos. La aplicación o el objeto que empieza a reproducir un sonido definen el entorno limitado de propietario.

El método `SoundMixer.stopAll()` detiene los sonidos de todos los objetos SoundChannel que se reproducen en ese momento, siempre y cuando cumplan los criterios siguientes:

- Los objetos del mismo entorno limitado de propietario han iniciado los sonidos.
- Los sonidos proceden de un origen con un archivo de política que proporciona acceso al dominio de la aplicación o del objeto que llama al método `SoundMixer.stopAll()`.

Sin embargo, en una aplicación de AIR, el contenido del entorno limitado de seguridad de la aplicación (contenido instalado con la aplicación AIR) no está limitado por estas restricciones de seguridad.

Para saber si el método `SoundMixer.stopAll()` detendrá realmente todos los sonidos en reproducción, la aplicación puede llamar al método `SoundMixer.areSoundsInaccessible()`. Si el método devuelve un valor `true`, algunos de los sonidos que se reproducen se encuentran fuera del control del entorno limitado de propietario actual y no se detendrán con el método `SoundMixer.stopAll()`.

El método `SoundMixer.stopAll()` también evita que la cabeza lectora continúe para todos los sonidos cargados desde archivos externos. Sin embargo, los sonidos que se incorporan en archivos FLA y se asocian a los fotogramas de la línea de tiempo con la herramienta de edición de Flash podrían volver a reproducirse si la animación se desplaza a un nuevo fotograma.

Control de desplazamiento y volumen de sonido

El objeto SoundChannel controla los canales estéreo izquierdo y derecho de un sonido. Si el sonido MP3 es monoaural, los canales estéreo izquierdo y derecho del objeto SoundChannel incluirán formas de onda idénticas.

Se puede conocer la amplitud de cada canal estéreo del objeto que se reproduce mediante las propiedades `leftPeak` y `rightPeak` del objeto SoundChannel. Estas propiedades muestran la amplitud máxima de la forma de onda del sonido; no representan el volumen real de la reproducción. El volumen real de la reproducción es una función de la amplitud de la onda de sonido y de los valores de volumen establecidos en el objeto SoundChannel y la clase SoundMixer.

La propiedad pan (desplazamiento) de un objeto SoundChannel puede utilizarse para especificar un nivel de volumen diferente para cada uno de los canales izquierdo y derecho durante la reproducción. Esta propiedad puede tener un valor que oscila entre -1 y 1, donde -1 significa que el canal izquierdo se reproduce al máximo volumen mientras que el canal derecho está en silencio y 1 significa que el canal derecho se reproduce al máximo volumen mientras que el canal izquierdo está en silencio. Los valores numéricos entre -1 y 1 establecen valores proporcionales para los valores de canal izquierdo y derecho, y un valor 0 implica que los dos canales se reproducen a un volumen medio y equilibrado.

En el ejemplo de código siguiente se crea un objeto SoundTransform con un valor de volumen de 0,6 y un valor de desplazamiento de -1 (volumen máximo en el canal izquierdo y sin volumen en el canal derecho). Pasa el objeto SoundTransform como parámetro al método `play()`, que aplica dicho objeto al nuevo objeto SoundChannel que se crea para controlar la reproducción.

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

Se puede modificar el volumen y el desplazamiento mientras se reproduce un sonido estableciendo las propiedades `pan` o `volume` de un objeto SoundTransform y aplicando dicho objeto como propiedad `soundTransform` de un objeto SoundChannel.

También se pueden establecer simultáneamente valores globales de volumen y desplazamiento con la propiedad `soundTransform` de la clase SoundMixer, como se muestra en el ejemplo siguiente:

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

Además, se puede utilizar un objeto SoundTransform para establecer los valores de volumen y desplazamiento de un objeto Microphone (véase “Captura de entradas de sonido” en la página 599) y de los objetos Sprite y SimpleButton.

En el ejemplo siguiente se alterna el desplazamiento del sonido del canal izquierdo al derecho, y a la inversa, mientras se reproduce el sonido.

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;

var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
    trans.pan = Math.sin(panCounter);
    channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
    panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Este código se inicia cargando un archivo de sonido y creando un nuevo objeto SoundTransform con el volumen establecido en 1 (volumen máximo) y el desplazamiento establecido en 0 (equilibrio entre los canales izquierdo y derecho). A continuación, llama al método `snd.play()` y pasa el objeto SoundTransform como parámetro.

Mientras se reproduce el sonido, el método `onEnterFrame()` se ejecuta repetidamente. El método `onEnterFrame()` utiliza la función `Math.sin()` para generar un valor entre -1 y 1, rango que corresponde a los valores aceptables de la propiedad `SoundTransform.pan`. La propiedad `pan` del objeto SoundTransform se establece en el nuevo valor y , posteriormente, se establece la propiedad `soundTransform` del canal para utilizar el objeto SoundTransform alterado.

Para ejecutar este ejemplo, reemplace el nombre de archivo bigSound.mp3 con el nombre de un archivo MP3 local. A continuación, ejecute el ejemplo. Escuchará el aumento del volumen del canal izquierdo mientras disminuye el volumen del canal derecho, y viceversa.

En este ejemplo podría lograrse el mismo efecto estableciendo la propiedad `soundTransform` de la clase SoundMixer. Sin embargo, se vería afectado el desplazamiento de todos los sonidos en reproducción, no solo el sonido que reproduce este objeto SoundChannel.

Trabajo con metadatos de sonido

Los archivos de sonido que utilizan el formato MP3 pueden contener datos adicionales sobre el sonido en forma de etiquetas ID3.

No todos los archivos MP3 contienen metadatos ID3. Cuando un objeto Sound carga un archivo de sonido MP3, distribuye un evento `Event.ID3` si el archivo de sonido incluye metadatos ID3. Para evitar errores en tiempo de ejecución, la aplicación debe esperar a recibir el evento `Event.ID3` antes de acceder a la propiedad `Sound.id3` de un sonido cargado.

En el código siguiente se muestra la manera de detectar que se han cargado los metadatos ID3 para un archivo de sonido.

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
    var id3:ID3Info = event.target.id3;

    trace("Received ID3 Info:");
    for (var propName:String in id3)
    {
        trace(propName + " = " + id3[propName]);
    }
}
```

Este código empieza por crear un objeto Sound e indicarle que detecte el evento `Event.ID3`. Cuando se cargan los metadatos ID3 del archivo de sonido, se llama al método `onID3InfoReceived()`. El destino del objeto Event que se pasa al método `onID3InfoReceived()` es el objeto Sound original, de modo que el método obtiene la propiedad `id3` del objeto Sound y luego recorre todas sus propiedades con nombre para trazar sus valores.

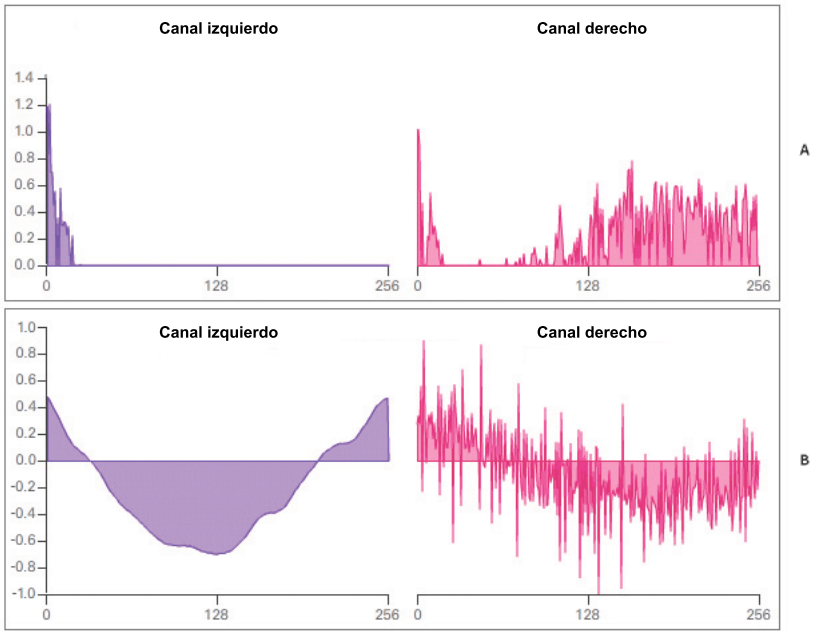
Acceso a datos de sonido sin formato

El método `SoundMixer.computeSpectrum()` permite que una aplicación lea los datos de sonido sin formato para la forma de onda que se reproduce en ese momento. Si se reproduce más de un objeto SoundChannel, el método `SoundMixer.computeSpectrum()` muestra los datos de sonido combinados de cada objeto SoundChannel mezclado.

Los datos de sonido se devuelven como un objeto ByteArray con 512 bytes de datos, cada uno de los cuales contiene un valor de coma flotante que oscila entre -1 y 1. Estos valores representan la amplitud de los puntos de la forma de onda del sonido en reproducción. Los valores se transmiten en dos grupos de 256; el primer grupo para el canal estéreo izquierdo y el segundo para el canal estéreo derecho.

El método `SoundMixer.computeSpectrum()` devolverá datos del espectro de frecuencias en lugar de datos de forma de onda si el parámetro `FFTMode` se establece en `true`. El espectro de frecuencias muestra la amplitud clasificada por la frecuencia de sonido, de menor a mayor. El algoritmo FFT (Fast Fourier Transform) se utiliza para convertir los datos de forma de onda en datos del espectro de frecuencias. Los valores resultantes del espectro de frecuencias oscilan entre 0 y 1,414 aproximadamente (la raíz cuadrada de 2).

En el siguiente diagrama se comparan los datos devueltos por el método `computeSpectrum()` cuando el parámetro `FFTMode` se establece en `true` y cuando se establece en `false`. El sonido cuyos datos se han utilizado para este diagrama contiene un sonido alto de contrabajo en el canal izquierdo y un sonido de batería en el canal derecho.



Valores devueltos por el método SoundMixer.computeSpectrum()
***A.** fftMode=true **B.** fftMode=false*

El método `computeSpectrum()` también puede devolver datos que se han vuelto a muestrear a una velocidad inferior. Normalmente, esto genera datos de forma de onda o de frecuencia más suaves a expensas del detalle. El parámetro `stretchFactor` controla la velocidad a la que se muestrea el método `computeSpectrum()`. Cuando el parámetro `stretchFactor` se establece en 0 (valor predeterminado), los datos de sonido se muestrean a una velocidad de 44,1 kHz. La velocidad se reduce a la mitad en los valores sucesivos del parámetro stretchFactor, de manera que un valor de 1 especifica una velocidad de 22,05 kHz, un valor de 2 especifica una velocidad de 11,025 kHz, etc. El método `computeSpectrum()` aún devuelve 256 bytes en cada canal estéreo cuando se utiliza un valor `stretchFactor` más alto.

El método `SoundMixer.computeSpectrum()` tiene algunas limitaciones:

- Puesto que los datos de sonido de un micrófono o de los flujos RTMP no pasan por el objeto SoundMixer global, el método `SoundMixer.computeSpectrum()` no devolverá datos de dichas fuentes.
- Si uno o varios sonidos que se reproducen proceden de orígenes situados fuera del entorno limitado de contenido actual, las restricciones de seguridad harán que el método `SoundMixer.computeSpectrum()` genere un error. Para obtener más detalles sobre las limitaciones de seguridad del método `SoundMixer.computeSpectrum()`, consulte "Consideraciones de seguridad al cargar y reproducir sonidos" en la página 592 y "Acceso a medios cargados como datos" en la página 734.

Sin embargo, en una aplicación de AIR, el contenido del entorno limitado de seguridad de la aplicación (contenido instalado con la aplicación AIR) no está limitado por estas restricciones de seguridad.

Creación de un visualizador de sonido sencillo

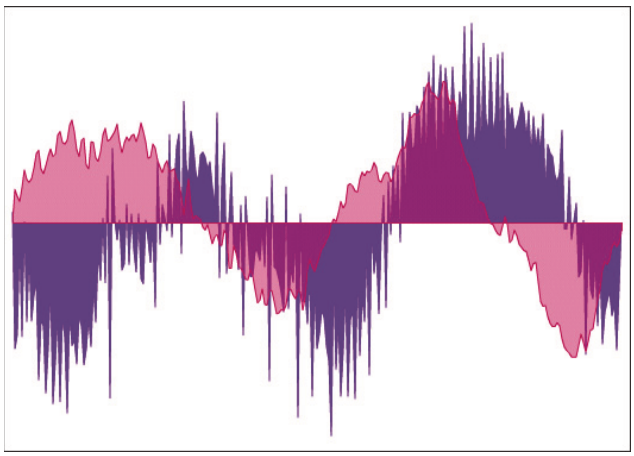
En el ejemplo siguiente se utiliza el método `SoundMixer.computeSpectrum()` para mostrar un gráfico de la forma de onda que se anima con cada fotograma:

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();

function onEnterFrame(event:Event):void
{
    SoundMixer.computeSpectrum(bytes, false, 0);

    var g:Graphics = this.graphics;

    g.clear();
    g.lineStyle(0, 0x6600CC);
    g.beginFill(0x6600CC);
    g.moveTo(0, PLOT_HEIGHT);

    var n:Number = 0;

    // left channel
    for (var i:int = 0; i < CHANNEL_LENGTH; i++)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
```

```
    g.endFill();

    // right channel
    g.lineStyle(0, 0xCC0066);
    g.beginFill(0xCC0066, 0.5);
    g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

    for (i = CHANNEL_LENGTH; i > 0; i--)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(0, PLOT_HEIGHT);
    g.endFill();
}

function onPlaybackComplete(event:Event)
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

En primer lugar, este ejemplo carga y reproduce un archivo de sonido, y luego detecta el evento `Event.ENTER_FRAME` que activará el método `onEnterFrame()` mientras se reproduce el sonido. El método `onEnterFrame()` se inicia llamando al método `SoundMixer.computeSpectrum()`, que almacena los datos de forma de onda del sonido en el objeto ByteArray `bytes`.

La forma de onda del sonido se representa con la API de dibujo vectorial. El bucle `for` recorre los primeros 256 valores de datos; representa el canal estéreo izquierdo y dibuja una línea desde un punto al siguiente con el método `Graphics.lineTo()`. Un segundo bucle `for` recorre el siguiente conjunto de 256 valores y los representa en orden inverso esta vez, de derecha a izquierda. Las representaciones de la forma de onda resultante pueden generar un interesante efecto de reflejo de imagen, tal como se muestra en la imagen siguiente.



Captura de entradas de sonido

La clase Microphone permite a la aplicación conectar un micrófono u otro dispositivo de entrada de sonido en el sistema del usuario, y difundir el audio de la entrada a los altavoces, o bien enviar los datos de audio a un servidor remoto, como Flash Media Server. No se puede acceder a los datos de audio sin procesar desde el micrófono; sólo se puede enviar audio a los altavoces del sistema o enviar datos de audio comprimidos a un servidor remoto. Se puede emplear el códec Speex o Nellymoser para los datos enviados a un servidor remoto. (El códec Speex se admite a partir de Flash Player 10 y Adobe AIR 1.5.)

Acceso a un micrófono

La clase Microphone no tiene un método constructor. En su lugar, se utiliza el método `Microphone.getMicrophone()` estático para obtener una nueva instancia de Microphone, tal como se muestra a continuación.

```
var mic:Microphone = Microphone.getMicrophone();
```

Si se llama al método `Microphone.getMicrophone()` sin un parámetro, se devuelve el primer dispositivo de entrada de sonido detectado en el sistema del usuario.

Un sistema puede tener más de un dispositivo de entrada de sonido conectado. La aplicación puede utilizar la propiedad `Microphone.names` para obtener un conjunto de los nombres de todos los dispositivos de entrada de sonido disponibles. A continuación, puede llamar al método `Microphone.getMicrophone()` con un parámetro `index` que coincide con el valor de índice del nombre de un dispositivo del conjunto.

Es posible que el sistema no tenga conectado un micrófono u otro dispositivo de entrada de sonido. Se puede utilizar la propiedad `Microphone.names` o el método `Microphone.getMicrophone()` para comprobar si el usuario tiene un dispositivo de entrada de sonido instalado. Si el usuario no tiene un dispositivo de entrada de sonido instalado, la longitud del conjunto `names` es cero y el método `getMicrophone()` devuelve un valor `null`.

Cuando la aplicación llama al método `Microphone.getMicrophone()`, Flash Player muestra el cuadro de diálogo Configuración de Flash Player, que pregunta al usuario si desea que Flash Player acceda a la cámara y al micrófono del sistema. Una vez que el usuario hace clic en el botón Allow (Permitir) o en el botón Deny (Denegar) de este cuadro de diálogo, se distribuye un objeto StatusEvent. La propiedad `code` de la instancia de StatusEvent indica si se ha permitido o denegado el acceso al micrófono, tal como se muestra en este ejemplo:

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
    if (event.code == "Microphone.Unmuted")
    {
        trace("Microphone access was allowed.");
    }
    else if (event.code == "Microphone.Muted")
    {
        trace("Microphone access was denied.");
    }
}
```

La propiedad `StatusEvent.code` contiene "Microphone.Unmuted" si se ha permitido el acceso, o bien "Microphone.Muted" si se ha denegado.

Nota: *la propiedad* `Microphone.muted` *se establece en* `true` *o* `false` *cuando el usuario permite o deniega el acceso al micrófono, respectivamente. Sin embargo, la propiedad* `muted` *no se establece en la instancia de Microphone hasta que se haya distribuido el objeto StatusEvent, de manera que la aplicación también debe esperar a que el evento* `StatusEvent.STATUS` *se distribuya antes de comprobar la propiedad* `Microphone.muted`*.*

Enrutamiento de audio de micrófono a altavoces locales

La entrada de audio de un micrófono puede enrutarse a los altavoces del sistema locales llamando al método `Microphone.setLoopback()` con un valor de parámetro `true`.

Cuando el sonido de un micrófono local se enruta a los altavoces locales, existe el riesgo de crear un bucle de acoplamiento de audio, que puede provocar sonidos estridentes e incluso dañar el hardware de sonido. Si se llama al método `Microphone.setUseEchoSuppression()` con un valor de parámetro `true`, se reduce (sin eliminarse por completo) el riesgo de que se produzca dicho acoplamiento de audio. Adobe recomienda llamar siempre a `Microphone.setUseEchoSuppression(true)` antes de llamar a `Microphone.setLoopback(true)`, a menos que se sepa con certeza que el usuario reproduce el sonido con auriculares u otro dispositivo distinto de los altavoces.

En el código siguiente se muestra cómo enrutar el audio de un micrófono local a los altavoces del sistema locales:

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

Alteración del audio de micrófono

La aplicación puede alterar los datos de audio que proceden de un micrófono de dos maneras. Primero, puede cambiar la ganancia del sonido de la entrada, que multiplica de forma efectiva los valores de entrada por una cantidad especificada para crear un sonido más alto o más bajo. La propiedad `Microphone.gain` acepta valores numéricos entre 0 y 100 (ambos incluidos). Un valor de 50 funciona como un multiplicador de 1 y especifica un volumen normal. Un valor de 0 funciona como un multiplicador de 0 y silencia de forma efectiva el audio de la entrada. Los valores superiores a 50 especifican un volumen por encima del normal.

La aplicación también puede cambiar la frecuencia de muestreo del audio de la entrada. Las frecuencias de muestreo más altas aumentan la calidad del sonido, pero crean flujos de datos más densos que utilizan más recursos para la transmisión y el almacenamiento. La propiedad `Microphone.rate` representa la frecuencia de muestreo de audio calculada en kilohercios (kHz). La frecuencia de muestreo predeterminada es de 8 kHz. La propiedad `Microphone.rate` puede establecerse en un valor mayor que 8 kHz si el micrófono admite una frecuencia superior. Por ejemplo, si se establece la propiedad `Microphone.rate` en un valor de 11, se establece la velocidad de muestreo en 11 kHz; si se establece en 22, se establece la velocidad de muestreo en 22 kHz, y así sucesivamente. Las frecuencias de muestreo disponibles dependen del códec seleccionado. Cuando se usa el códec Nellymoser, puede especificar 5, 8, 11, 16, 22 y 44 kHz como frecuencia de muestreo. Si se utiliza el códec Speex (disponible a partir de Flash Player 10 y Adobe AIR 1.5), sólo es posible utilizar frecuencias de 16 kHz.

Detección de actividad del micrófono

Para conservar los recursos de procesamiento y ancho de banda, Flash Player intenta detectar el momento en que el micrófono no transmite ningún sonido. Cuando el nivel de actividad del micrófono se mantiene por debajo del umbral de nivel de silencio durante un período de tiempo, Flash Player deja de transmitir la entrada de audio y distribuye un objeto ActivityEvent en su lugar. Si se utiliza el códec Speex (disponible en Flash Player 10 y en Adobe AIR 1.5 o versiones posteriores), se debe establecer el nivel de silencio en 0 para garantizar que la aplicación transmita datos de audio de forma continuada. La detección de actividad de voz de Speex reduce automáticamente el ancho de banda.

La clase Microphone tiene tres propiedades para supervisar y controlar la detección de actividad:

- La propiedad de sólo lectura `activityLevel` indica la cantidad de sonido que detecta el micrófono, en una escala de 0 a 100.
- La propiedad `silenceLevel` especifica la cantidad de sonido necesaria para activar el micrófono y distribuir un evento `ActivityEvent.ACTIVITY`. La propiedad `silenceLevel` también utiliza una escala de 0 a 100, y el valor predeterminado es 10.
- La propiedad `silenceTimeout` describe el número de milisegundos durante los cuales el nivel de actividad debe permanecer por debajo del nivel de silencio, hasta que se distribuye un evento `ActivityEvent.ACTIVITY` para indicar que el micrófono ya está en silencio. El valor predeterminado de `silenceTimeout` es 2000.

Las propiedades `Microphone.silenceLevel` y `Microphone.silenceTimeout` son de sólo lectura, pero se puede utilizar el método `Microphone.setSilenceLevel()` para cambiar sus valores.

En algunos casos, el proceso de activación del micrófono cuando se detecta una nueva actividad puede provocar una breve demora. Esas demoras de activación pueden eliminarse manteniendo el micrófono activo todo el tiempo. La aplicación puede llamar al método `Microphone.setSilenceLevel()` con el parámetro `silenceLevel` establecido en cero para indicar a Flash Player que mantenga el micrófono activo y siga recopilando datos de audio, incluso cuando no se detecten sonidos. Y a la inversa, si se establece el parámetro `silenceLevel` en 100, se evita que el micrófono esté siempre activado.

El ejemplo siguiente muestra información sobre el micrófono e informa de los eventos de actividad y de estado que distribuye un objeto Microphone:

```
import flash,events.ActivityEvent;
import flash,events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
    trace(" " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);
```

```
var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);

function onMicActivity(event:ActivityEvent):void
{
    trace("activating=" + event.activating + ", activityLevel=" +
        mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
    trace("status: level=" + event.level + ", code=" + event.code);
}
```

Cuando ejecute el ejemplo anterior, hable al micrófono del sistema o haga ruido, y verá que aparecen las sentencias trace resultantes en una consola o una ventana de depuración.

Intercambio de audio con un servidor multimedia

Al utilizar ActionScript con un servidor de flujo de medios como Flash Media Server estarán disponibles otras funciones de audio.

En concreto, la aplicación puede asociar un objeto Microphone a un objeto NetStream y transmitir datos directamente desde el micrófono del usuario al servidor. Los datos de audio también se pueden transmitir desde el servidor a una aplicación creada con Flash o Flex y reproducirse como parte de un objeto MovieClip, o bien mediante un objeto Video.

El códec Speex está disponible a partir de Flash Player 10 y Adobe AIR 1.5. Para definir el códec utilizado para audio comprimido enviado al servidor de medios, se debe establecer la propiedad `codec` del objeto Microphone. Esta propiedad puede tener dos valores, que se enumeran en la clase SoundCodec. Con la definición de la propiedad codec como `SoundCodec.SPEEX`, se selecciona el códec Speex para comprimir audio. Con el establecimiento de la propiedad en `SoundCodec.NELLYMOSER` (valor predeterminado), se selecciona el códec Nellymoser para comprimir audio.

Para obtener más información, consulte la documentación en línea de Flash Media Server en http://livedocs.adobe.com.

Ejemplo: Podcast Player

Una emisión podcast es un archivo de sonido que se distribuye por Internet, bajo demanda o por suscripción. Las emisiones podcast suelen publicarse como parte de una serie (denominada "canal podcast"). Puesto que los episodios de emisiones podcast pueden durar entre un minuto y muchas horas, normalmente se transmiten mientras se reproducen. Los episodios de emisiones podcast, también denominados "elementos", se suelen transmitir en formato MP3. Las emisiones podcast de vídeo gozan de una gran popularidad, pero esta aplicación de ejemplo sólo reproduce emisiones podcast de audio que utilizan archivos MP3.

Este ejemplo no constituye un agregador de emisiones podcast completo. Por ejemplo, no gestiona suscripciones a emisiones podcast específicas ni recuerda las emisiones podcast que ha escuchado el usuario la próxima vez que se ejecuta la aplicación. Podría servir de punto de partida para un agregador de emisiones podcast más completo.

El ejemplo de Podcast Player ilustra las siguientes técnicas de programación con ActionScript:

- Leer un fuente RSS externa y analizar su contenido XML
- Crear una clase SoundFacade para simplificar la carga y la reproducción de archivos de sonido
- Mostrar el progreso de la reproducción de sonido
- Pausar y reanudar la reproducción de sonido

Para obtener los archivos de la aplicación para esta muestra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Podcast Player se encuentran en la carpeta Samples/PodcastPlayer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
PodcastPlayer.mxml o PodcastPlayer.fla	La interfaz de usuario de la aplicación para Flex (MXML) o Flash (FLA).
comp/example/programmingas3/podcastplayer/PodcastPlayer.as	Clase de documento que contiene la lógica de la interfaz de usuario para el reproductor de podcast (sólo Flash).
comp/example/programmingas3/podcastplayer/SoundPlayer.as	Clase del símbolo de clip de película SoundPlayer que contiene la lógica de la interfaz de usuario para el reproductor de sonidos (sólo Flash).
comp/example/programmingas3/podcastplayer/PlayButtonRenderer.as	Procesador de celdas personalizado para visualizar un botón de reproducción en una celda de cuadrícula de datos (sólo Flash).
com/example/programmingas3/podcastplayer/RSSBase.as	Una clase base que proporciona propiedades y métodos comunes para las clases RSSChannel y RSSItem.
com/example/programmingas3/podcastplayer/RSSChannel.as	Una clase ActionScript que contiene datos sobre un canal RSS.
com/example/programmingas3/podcastplayer/RSSItem.as	Una clase ActionScript que contiene datos sobre un elemento RSS.
com/example/programmingas3/podcastplayer/SoundFacade.as	La clase ActionScript principal de la aplicación. Reúne los métodos y los eventos de las clases Sound y SoundChannel, y añade las funciones de pausa y reanudación de reproducción.
com/example/programmingas3/podcastplayer/URLService.as	Una clase ActionScript que recupera datos de un URL remoto.
playerconfig.xml	Un archivo XML que contiene una lista de las fuentes RSS que representan canales podcast.
comp/example/programmingas3/utils/DateUtil.as	Clase que se utiliza para aplicar formato de fechas fácilmente (sólo Flash).

Lectura de datos RSS para un canal podcast

La aplicación Podcast Player empieza leyendo información sobre varios canales podcast y sus episodios:

1. En primer lugar, la aplicación lee un archivo de configuración XML que contiene una lista de canales podcast y muestra dicha lista al usuario.

2. Cuando el usuario selecciona uno de los canales podcast, lee la fuente RSS del canal y muestra una lista de los episodios del canal.

Este ejemplo utiliza la clase de utilidad URLLoader para recuperar datos basados en texto desde una ubicación remota o un archivo local. Podcast Player crea primero un objeto URLLoader para obtener una lista de fuentes RSS en formato XML del archivo playerconfig.xml. A continuación, cuando el usuario selecciona una fuente específica de la lista, se crea un nuevo objeto URLLoader para leer datos RSS del URL de dicha fuente.

Simplificación de la carga y la reproducción de sonido mediante la clase SoundFacade

La arquitectura de sonido ActionScript 3.0 es eficaz, pero compleja. Las aplicaciones que sólo necesitan funciones básicas de carga y reproducción de sonido pueden utilizar una clase que oculta parte de la complejidad; dicha clase proporciona un conjunto más sencillo de llamadas de método y eventos. En el mundo de los patrones de diseño de software, esta clase se denomina *facade*.

La clase SoundFacade presenta una interfaz única para realizar las tareas siguientes:

- Cargar archivos de sonido con un objeto Sound, un objeto SoundLoaderContext y la clase SoundMixer
- Reproducir archivos de sonido con los objetos Sound y SoundChannel
- Distribuir eventos de progreso de reproducción
- Pausar y reanudar la reproducción del sonido mediante los objetos Sound y SoundChannel

La clase SoundFacade intenta ofrecer la mayor parte de las funciones de las clase de sonido ActionScript, pero con una menor complejidad.

En el código siguiente se muestra la declaración de clase, las propiedades de clase y el método constructor `SoundFacade()`:

```
public class SoundFacade extends EventDispatcher
{
    public var s:Sound;
    public var sc:SoundChannel;
    public var url:String;
    public var bufferTime:int = 1000;

    public var isLoaded:Boolean = false;
    public var isReadyToPlay:Boolean = false;
    public var isPlaying:Boolean = false;
    public var isStreaming:Boolean = true;
    public var autoLoad:Boolean = true;
    public var autoPlay:Boolean = true;

    public var pausePosition:int = 0;

    public static const PLAY_PROGRESS:String = "playProgress";
    public var progressInterval:int = 1000;
    public var playTimer:Timer;
```

```
public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                            autoPlay:Boolean = true, streaming:Boolean = true,
                            bufferTime:int = -1):void
{
    this.url = soundUrl;

    // Sets Boolean values that determine the behavior of this object
    this.autoLoad = autoLoad;
    this.autoPlay = autoPlay;
    this.isStreaming = streaming;

    // Defaults to the global bufferTime value
    if (bufferTime < 0)
    {
        bufferTime = SoundMixer.bufferTime;
    }

    // Keeps buffer time reasonable, between 0 and 30 seconds
    this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

    if (autoLoad)
    {
        load();
    }
}
```

La clase SoundFacade amplía la clase EventDispatcher para que pueda distribuir sus propios eventos. El código de clase declara primero las propiedades de un objeto Sound y un objeto SoundChannel. La clase también almacena el valor del URL del archivo de sonido y una propiedad `bufferTime` que se utilizará al transmitir el sonido. Asimismo, acepta algunos valores de parámetro booleanos que afectan al comportamiento de carga y reproducción.

- El parámetro `autoLoad` indica al objeto que la carga de sonido debe empezar en cuanto se cree este objeto.
- El parámetro `autoPlay` indica que la reproducción de sonido debe empezar en cuanto se hayan cargado suficientes datos de sonido. Si se trata de un flujo de sonido, la reproducción empezará tan pronto como se hayan cargado suficientes datos, según especifica la propiedad `bufferTime`.
- El parámetro `streaming` indica que este archivo de sonido puede empezar a reproducirse antes de que haya finalizado la carga.

El parámetro `bufferTime` se establece de forma predeterminada en un valor de -1. Si el método constructor detecta un valor negativo en el parámetro `bufferTime`, establece la propiedad `bufferTime` en el valor de `SoundMixer.bufferTime`. Esto permite que la aplicación se establezca de forma predeterminada en el valor `SoundMixer.bufferTime` global, según se desee.

Si el parámetro `autoLoad` se establece en `true`, el método constructor llama inmediatamente al siguiente método `load()` para iniciar la carga del archivo de sonido:

```
public function load():void
{
    if (this.isPlaying)
    {
        this.stop();
        this.s.close();
    }
    this.isLoaded = false;

    this.s = new Sound();

    this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
    this.s.addEventListener(Event.OPEN, onLoadOpen);
    this.s.addEventListener(Event.COMPLETE, onLoadComplete);
    this.s.addEventListener(Event.ID3, onID3);
    this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
    this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

    var req:URLRequest = new URLRequest(this.url);

    var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime, true);
    this.s.load(req, context);
}
```

El método `load()` crea un nuevo objeto Sound y añade detectores para todos los eventos de sonido importantes. A continuación, indica al objeto Sound que cargue el archivo de sonido, mediante un objeto SoundLoaderContext para pasar el valor `bufferTime`.

Puesto que se puede cambiar la propiedad `url`, se puede usar una instancia SoundFacade para reproducir distintos archivos de sonido en secuencia: sólo hay que cambiar la propiedad `url` y llamar al método `load()` para que se cargue el nuevo archivo de sonido.

Los tres siguientes métodos de detección de eventos muestran la forma en que el objeto SoundFacade hace un seguimiento del progreso de carga y decide el momento en que se inicia la reproducción del sonido:

```
public function onLoadOpen(event:Event):void
{
    if (this.isStreaming)
    {
        this.isReadyToPlay = true;
        if (autoPlay)
        {
            this.play();
        }
    }
    this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
    this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
    this.isReadyToPlay = true;
    this.isLoaded = true;
    this.dispatchEvent(evt.clone());

    if (autoPlay && !isPlaying)
    {
        play();
    }
}
```

El método `onLoadOpen()` se ejecuta cuando se inicia la carga de sonido. Si se puede reproducir el sonido en el modo de transmisión, el método `onLoadComplete()` establece el indicador `isReadyToPlay` en `true` inmediatamente. El indicador `isReadyToPlay` determina si la aplicación puede empezar a reproducir el sonido, tal vez como respuesta a una acción de usuario, como hacer clic en un botón Reproducir. La clase SoundChannel administra el búfer de los datos de sonido, de modo que no es necesario comprobar explícitamente si se han cargado suficientes datos antes de llamar al método `play()`.

El método `onLoadProgress()` se ejecuta periódicamente durante el proceso de carga. Distribuye un clon de su objeto ProgressEvent que utilizará el código que usa este objeto SoundFacade.

Cuando los datos de sonido se han cargado completamente, se ejecuta el método `onLoadComplete()` y se llama al método `play()` para sonidos que no sean de una transmisión de flujo si es necesario. El método `play()` se muestra a continuación.

```
public function play(pos:int = 0):void
{
    if (!this.isPlaying)
    {
        if (this.isReadyToPlay)
        {
            this.sc = this.s.play(pos);
            this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
            this.isPlaying = true;

            this.playTimer = new Timer(this.progressInterval);
            this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
            this.playTimer.start();
        }
    }
}
```

El método `play()` llama al método `Sound.play()` si el sonido está listo para reproducirse. El objeto SoundChannel resultante se almacena en la propiedad `sc`. El método `play()` crea un objeto Timer que se utilizará para distribuir eventos de progreso de reproducción a intervalos regulares.

Visualización del progreso de reproducción

Crear un objeto Timer para controlar la reproducción es una operación compleja que sólo se debería tener que programar una sola vez. La encapsulación de esta lógica Timer en una clase reutilizable (como la clase SoundFacade) permite que las aplicaciones detecten las mismas clases de eventos de progreso cuando se carga y se reproduce un sonido.

El objeto Timer que crea el método `SoundFacade.play()` distribuye una instancia de TimerEvent cada segundo. El siguiente método `onPlayTimer()` se ejecuta cuando llega una nueva clase TimerEvent.

```
public function onPlayTimer(event:TimerEvent):void
{
    var estimatedLength:int =
        Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
    var progEvent:ProgressEvent =
        new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position, estimatedLength);
    this.dispatchEvent(progEvent);
}
```

El método `onPlayTimer()` implementa la técnica de estimación de tamaño descrita en la sección"Control de la reproducción" en la página 589. Luego crea una nueva instancia de ProgressEvent con un tipo de evento de `SoundFacade.PLAY_PROGRESS`, con la propiedad `bytesLoaded` establecida en la posición actual del objeto SoundChannel y la propiedad `bytesTotal` establecida en la duración estimada de los datos de sonido.

Pausa y reanudación de la reproducción

El método `SoundFacade.play()` mostrado anteriormente acepta un parámetro `pos` correspondiente a un punto inicial de los datos de sonido. Si el valor `pos` es cero, el sonido empieza a reproducirse desde el principio.

El método `SoundFacade.stop()` también acepta un parámetro `pos`, tal como se muestra aquí:

```
public function stop(pos:int = 0):void
{
    if (this.isPlaying)
    {
        this.pausePosition = pos;
        this.sc.stop();
        this.playTimer.stop();
        this.isPlaying = false;
    }
}
```

Cuando se llama al método `SoundFacade.stop()`, éste establece la propiedad `pausePosition` para que la aplicación sepa dónde colocar la cabeza lectora si el usuario desea reanudar la reproducción del mismo sonido.

Los métodos `SoundFacade.pause()` y `SoundFacade.resume()` que se muestran a continuación invocan los métodos `SoundFacade.stop()` y `SoundFacade.play()` respectivamente y pasan un valor de parámetro `pos` cada vez.

```
public function pause():void
{
    stop(this.sc.position);
}

public function resume():void
{
    play(this.pausePosition);
}
```

El método `pause()` pasa el valor `SoundChannel.position` al método `play()`, que almacena dicho valor en la propiedad `pausePosition`. El método `resume()` empieza a reproducir de nuevo el mismo sonido con el valor `pausePosition` como punto inicial.

Ampliación del ejemplo de Podcast Player

En este ejemplo se presenta un Podcast Player básico que aborda la utilización de la clase SoundFacade reutilizable. También se podrían añadir otras funciones para ampliar la utilidad de esta aplicación, entre las que se incluyen:

- Almacenar la lista de fuentes e información de utilización sobre cada episodio de una instancia de SharedObject que se pueda utilizar la próxima vez que el usuario ejecute la aplicación.
- Permitir que el usuario añada su propia fuente RSS a la lista de canales podcast.
- Recordar la posición de la cabeza lectora cuando el usuario detenga o abandone un episodio; de este modo, se puede reiniciar desde dicho punto la próxima vez que el usuario ejecute la aplicación.
- Descargar archivos MP3 de episodios para escuchar cuando el usuario no esté conectado a Internet.
- añadir funciones de suscripción que busquen nuevos episodios periódicamente en un canal podcast y actualicen la lista de episodios de forma automática.
- Añadir funciones de búsqueda y exploración de emisiones podcast con la API de un servicio de alojamiento de podcast, como Odeo.com.

Capítulo 26: Captura de entradas del usuario

En este capítulo se describe la manera de crear interactividad mediante ActionScript 3.0 para responder a las acciones del usuario. Primero se describen los eventos de teclado y de ratón, y después se pasa a temas más avanzados, como la personalización de un menú contextual y la administración de la selección. El capítulo concluye con WordSearch, un ejemplo de aplicación que responde a entradas del ratón.

Para entender este capítulo hay que conocer el modelo de eventos de ActionScript 3.0. Para obtener más información, consulte “Gestión de eventos” en la página 254.

Fundamentos de la captura de entradas del usuario

Introducción a la captura de entradas de usuario

La interacción con el usuario, ya sea con el teclado, el ratón, la cámara o una combinación de estos dispositivos, es la base de interactividad. En ActionScript 3.0, identificar la interacción del usuario y responder a la misma requieren básicamente detectar eventos.

La clase InteractiveObject, una subclase de la clase DisplayObject, proporciona la estructura común de eventos y la funcionalidad necesaria para controlar la interacción con el usuario. Las instancias de la clase InteractiveObject no se crean directamente. Los objetos de visualización, como SimpleButton, Sprite, TextField y diversos componentes de Flex y la herramienta de edición de Flash heredan su modelo de interacción con el usuario de esta clase y, por tanto, comparten una estructura común. Esto significa que las técnicas que aprenda y el código que escriba para controlar la interacción del usuario en un objeto derivado de InteractiveObject son aplicables a todos los demás.

En este capítulo se describen las siguientes tareas comunes de interacción con el usuario:

- Capturar entradas del teclado en toda la aplicación
- Capturar entradas del teclado en un objeto de visualización específico
- Capturar acciones del ratón en toda la aplicación
- Capturar entradas del ratón en un objeto de visualización específico
- Crear interactividad de arrastrar y colocar
- Crear un cursor de ratón personalizado (puntero del ratón)
- Añadir nuevos comportamientos al menú contextual
- Administración de la selección

Conceptos y términos importantes

Antes de continuar, es importante familiarizarse con los siguientes términos clave relacionados con la interacción del usuario:

- Código de caracteres: código numérico que representa un carácter del juego de caracteres actual (asociado con la pulsación de una tecla en el teclado). Por ejemplo, "D" y "d" tienen distintos códigos de carácter, aunque se crean con la misma tecla en un teclado inglés de EE.UU.

- Menú contextual: menú que aparece cuando un usuario hace clic con el botón derecho o utiliza una combinación específica de teclado-ratón. Los comandos de menú contextual suelen aplicarse directamente al objeto en el que se ha hecho clic. Por ejemplo, un menú contextual para una imagen puede contener un comando para mostrar la imagen en una ventana independiente y un comando para descargarla.
- Selección: indicación de que un elemento seleccionado está activo y es el destino de una interacción de teclado o ratón.
- Código de tecla: código numérico correspondiente a una tecla física del teclado.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Como este capítulo se centra en trabajar con entradas del usuario en ActionScript, prácticamente todos los listados de código requieren manipular algún tipo de objeto de visualización, generalmente un campo de texto o cualquier subclase de InteractiveObject. Para estos ejemplos, el objeto de visualización puede ser un objeto creado y colocado en el escenario en Adobe® Flash® CS4 Professional o un objeto creado con ActionScript. Para probar un ejemplo el resultado debe verse en Flash Player o Adobe® AIR™ e interactuar con el ejemplo para ver los efectos del código.

Para probar los listados de código de este capítulo:

1. Cree un documento vacío utilizando la herramienta de edición de Flash.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Cree una instancia en el escenario:
   - Si el código hace referencia a un campo de texto, use la herramienta Texto para crear un campo de texto dinámico en el escenario.
   - De lo contrario, cree un botón o una instancia de símbolo de clip de película en el escenario.
5. Seleccione el campo de texto, el botón o el clip de película y asígnele un nombre de instancia en el inspector de propiedades. El nombre debe coincidir con el nombre del objeto de visualización del código de ejemplo; por ejemplo, si el código manipula un objeto denominado `myDisplayObject`, asigne al objeto Stage el nombre `myDisplayObject`.
6. Ejecute el programa seleccionando Control > Probar película.

   En la pantalla se manipula el objeto de visualización especificado en el código.

Captura de entradas de teclado

Los objetos de visualización que heredan su modelo de interacción de la clase InteractiveObject pueden responder a eventos del teclado mediante detectores de eventos. Por ejemplo, se puede colocar un detector de eventos en el escenario para detectar entradas de teclado y responder a las mismas. En el código siguiente, un detector de eventos captura una pulsación de tecla y se muestran las propiedades de nombre de tecla y código de tecla:

```
function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (character code: " +
event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Alguna teclas, como la tecla Ctrl, generan eventos aunque no tengan un glifo de representación.

En el ejemplo de código anterior, el detector de eventos del teclado captura la entrada de teclado para todo el escenario. También se puede escribir un detector de eventos para un objeto de visualización específico en el escenario; este detector de eventos se activaría cuando se seleccionara el objeto.

En el siguiente ejemplo, las pulsaciones de teclas se reflejan en el panel Salida cuando el usuario escribe en la instancia de TextField. Si se mantiene presionada la tecla Mayús, el color del borde de TextField cambiará temporalmente a rojo.

En este código se supone que hay una instancia de TextField denominada `tf` en el objeto Stage (el escenario).

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (key code: " +
event.keyCode + " character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
    trace("Key Released: " + String.fromCharCode(event.charCode) + " (key code: " +
event.keyCode + " character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT)
    {
        tf.borderColor = 0x000000;
    }
}
```

La clase TextField también notifica un evento `textInput` que se puede detectar cuando un usuario introduce texto. Para obtener más información, consulte “Captura de una entrada de texto” en la página 451.

Aspectos básicos de los códigos de teclas y los códigos de caracteres

Se puede acceder a las propiedades `keyCode` y `charCode` de un evento de teclado para determinar la tecla que se ha presionado y activar a continuación otras acciones. La propiedad `keyCode` es un valor numérico que se corresponde con el valor de una tecla del teclado. La propiedad `charCode` es el valor numérico de la tecla en el juego de caracteres actual. (El juego de caracteres predeterminado es UTF-8, que es compatible con ASCII.)

La principal diferencia entre el código de tecla y los valores de caracteres es que el valor del código de tecla representa una tecla determinada del teclado (el 1 de un teclado es distinto del 1 de la fila superior, pero la tecla que genera "1" y la tecla que genera "!" es la misma) y el valor de carácter representa un carácter determinado (los caracteres R y r son distintos).

Nota: *para obtener información sobre las asignaciones entre las teclas y sus valores de código de caracteres en ASCII, consulte la clase flash.ui.Keyboard en la Referencia del lenguaje ActionScript.*

Las correspondencias entre las teclas y sus códigos de tecla dependen del dispositivo y del sistema operativo. Por esta razón, no se deben utilizar asignaciones de teclas para activar acciones. Se deben utilizar los valores constantes predefinidos que proporciona la clase Keyboard para hacer referencia a las propiedades `keyCode` apropiadas. Por ejemplo, en lugar de utilizar la asignación de tecla para la tecla Mayús, se debe utilizar la constante `Keyboard.SHIFT` (como se indica en el ejemplo de código anterior).

Aspectos básicos de la precedencia de KeyboardEvent

Al igual que con los otros eventos, la secuencia de eventos de teclado se determina por la jerarquía de objetos de visualización, no en el orden en que se asignan métodos `addEventListener()` en el código.

Por ejemplo, supongamos que se coloca un campo de texto `tf` dentro de un clip de película denominado `container` y que se añade un detector de eventos para un evento de teclado en ambas instancias, como se muestra en el siguiente ejemplo:

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
    trace(event.currentTarget.name + " hears key press: " + String.fromCharCode(event.charCode)
+ " (key code: " + event.keyCode + " character code: " + event.charCode + ")");
}
```

Debido a que hay un detector en el campo de texto y en su contenedor principal, se llama dos veces a la función `reportKeyDown()` por cada pulsación en el campo de texto. Hay que tener en cuenta que por cada tecla pulsada, el campo de texto distribuye un evento antes de que el clip de película `contenedor` distribuya un evento.

El sistema operativo y el navegador Web procesarán los eventos de teclado antes que Adobe Flash Player o AIR. Por ejemplo, en Microsoft Internet Explorer, si se pulsan Ctrl+W se cierra la ventana del navegador antes de que un archivo SWF contenido pueda distribuir un evento de teclado.

Captura de entradas de ratón

Los clics del ratón crean eventos de ratón que pueden utilizarse para activar la funcionalidad de interacción. Se puede añadir un detector de eventos al objeto Stage para detectar los eventos de ratón que se produzcan en cualquier parte del archivo SWF. También se pueden añadir detectores de eventos a objetos del escenario que heredan de InteractiveObject (por ejemplo, Sprite o MovieClip); estos detectores se activan cuando se hace clic en el objeto.

Tal y como sucede con los eventos de teclado, los eventos de ratón se propagarán. En el siguiente ejemplo, dado que `square` es un objeto secundario de Stage, el evento se distribuirá tanto desde el objeto Sprite `square` como desde el objeto Stage cuando se haga clic en el cuadrado:

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
    trace(event.currentTarget.toString() + " dispatches MouseEvent. Local coords [" +
event.localX + "," + event.localY + "] Stage coords [" + event.stageX + "," + event.stageY + "]");
}
```

En el ejemplo anterior, el evento de ratón contiene información de posición sobre el clic. Las propiedades `localX` y `localY` contienen la ubicación donde tuvo lugar el clic en el objeto secundario inferior de la cadena de visualización. Por ejemplo, si se hace clic en la esquina superior izquierda de `square`, se notifican las coordenadas locales [0,0], ya que es el punto de registro de `square`. Como alternativa, las propiedades `stageX` y `stageY` hacen referencia a las coordenadas globales del clic en el escenario. El mismo clic notifica [50,50] para estas coordenadas, ya que `square` se movió a estas coordenadas. Ambos pares de coordenadas pueden ser útiles, dependiendo de cómo se desee responder a la interacción del usuario.

El objeto MouseEvent también contiene las propiedades booleanas `altKey`, `ctrlKey` y `shiftKey`. Se pueden utilizar estas propiedades para comprobar si también se está pulsando la tecla Alt, Ctrl o Mayús a la vez que se hace clic con el ratón.

Creación de funcionalidad de arrastrar y colocar

La funcionalidad de arrastrar y colocar permite a los usuarios seleccionar un objeto mientras se presiona el botón izquierdo del ratón, mover el objeto a una nueva ubicación en la pantalla y colocarlo en la nueva ubicación soltando el botón izquierdo del ratón. Esto se muestra en el siguiente ejemplo de código:

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);

var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
    circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
    circle.stopDrag();
    trace(circle.dropTarget.name);
}
```

Para obtener más información, consulte la sección que describe la creación de interacción de arrastrar y soltar en "Cambio de posición" en la página 300.

Personalización del cursor del ratón

El cursor (puntero) del ratón puede ocultarse o cambiarse por cualquier objeto de visualización en el escenario. Para ocultar el cursor del ratón, es necesario llamar al método `Mouse.hide()`. Para personalizar el cursor hay que llamar a `Mouse.hide()`, detectar el evento `MouseEvent.MOUSE_MOVE` en el objeto Stage y establecer las coordenadas de un objeto de visualización (el cursor personalizado) en las propiedades `stageX` y `stageY` del evento. El ejemplo siguiente muestra una ejecución básica de esta tarea:

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();

function redrawCursor(event:MouseEvent):void
{
    cursor.x = event.stageX;
    cursor.y = event.stageY;
}
```

Personalización del menú contextual

Cada objeto que hereda de la clase InteractiveObject tiene un menú contextual exclusivo, que se visualiza cuando un usuario hace clic con el botón derecho del ratón en el archivo SWF. Se incluyen varios comandos de manera predeterminada, incluidos Forward (Avanzar), Back (Atrás), Print (Imprimir), Quality (Calidad) y Zoom.

Se pueden eliminar todos los comandos predeterminados del menú, excepto Configuración y Acerca de. Si se establece la propiedad `showDefaultContextMenu` del objeto Stage en `false`, se eliminan estos comandos del menú contextual.

Para crear un menú contextual personalizado para un objeto de visualización específico hay que crear una nueva instancia de la clase ContextMenu, llamar al método `hideBuiltInItems()` y asignar dicha instancia a la propiedad `contextMenu` de la instancia de DisplayObject. El siguiente ejemplo proporciona un cuadrado dibujado dinámicamente con un comando de menú contextual para aplicarle un color aleatorio:

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
    square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    return new ColorTransform(Math.random(), Math.random(), Math.random(),1,(Math.random() *
512) - 255, (Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

Administración de la selección

Un objeto interactivo puede recibir la selección mediante programación o mediante una acción del usuario. En ambos casos, al establecer la selección se cambia el valor de la propiedad `focus` del objeto a `true`. Además, si la propiedad `tabEnabled` se establece en `true`, el usuario puede pasar la selección de un objeto a otro mediante la tecla Tabulador. Hay que tener en cuenta que el valor de `tabEnabled` es `false` de manera predeterminada, salvo en los siguientes casos:

- En un objeto SimpleButton, el valor es `true`.
- Para un campo de texto de entrada, el valor es `true`.
- En un objeto Sprite o MovieClip con `buttonMode` establecido en `true`, el valor es `true`.

En cada una de estas situaciones, se puede añadir un detector de `FocusEvent.FOCUS_IN` o `FocusEvent.FOCUS_OUT` para proporcionar un comportamiento adicional cuando cambie la selección. Esto es especialmente útil en los campos de texto y formularios, pero también se puede utilizar en objetos Sprite, MovieClip o en cualquier objeto que herede de la clase InteractiveObject. El siguiente ejemplo muestra cómo activar el cambio de selección de un objeto a otro con la tecla Tabulador y cómo responder al evento de selección posterior. En este caso, cada cuadrado cambia de color cuando se selecciona.

Nota: *la herramienta de edición Flash utiliza métodos abreviados de teclado para administrar la selección; por lo tanto, para simular correctamente la administración de la selección, hay que probar los archivos SWF en un navegador o AIR en lugar de en Flash.*

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
    for (j = 0; j < cols; j++)
    {
        createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
    }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
    var square:Sprite = new Sprite();
    square.graphics.beginFill(0x000000);
    square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
    square.graphics.endFill();
    square.x = startX;
    square.y = startY;
    square.tabEnabled = true;
    square.tabIndex = tabNumber;
    square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
    addChild(square);
}
function changeColor(event:FocusEvent):void
{
    event.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Ejemplo: WordSearch

En este ejemplo se ilustra la interacción del usuario mediante la gestión de eventos del ratón. Los usuarios deben crear la mayor cantidad posible de palabras a partir de una cuadrícula aleatoria de letras moviéndose horizontalmente o verticalmente por la cuadrícula, pero no pueden utilizar dos veces una misma letra. Este ejemplo muestra las siguientes características de ActionScript 3.0:

- Generación de una cuadrícula de componentes de forma dinámica
- Respuesta a eventos de ratón
- Mantenimiento de una puntuación según la interacción del usuario

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación WordSearch se encuentran en la carpeta Samples/WordSearch. La aplicación consta de los siguientes archivos:

Archivo	Descripción
WordSearch.as	La clase que proporciona la funcionalidad principal de la aplicación.
WordSearch.fla o WordSearch.mxml	Archivo principal de la aplicación de Flex (MXML) o de Flash (FLA).
dictionary.txt	Un archivo que se utiliza para determinar si las palabras escritas puntúan y están bien escritas.

Carga de un diccionario

Para crear un juego consistente en encontrar palabras hay que utilizar un diccionario. En el ejemplo se incluye un archivo de texto denominado dictionary.txt, que contiene una lista de palabras separadas por retornos de carro. Después de crear un conjunto denominado `words`, la función `loadDictionary()` solicita este archivo y, una vez cargado correctamente, lo convierte en una cadena larga. Esta cadena se puede analizar para generar un conjunto de palabras con el método `split()`, dividiéndola en cada instancia del retorno de carro (código de carácter 10) o línea nueva (código de carácter 13). Este análisis se realiza en la función `dictionaryLoaded()`:

```
words = dictionaryText.split(String.fromCharCode(13, 10));
```

Creación de la interfaz de usuario

Una vez almacenadas las palabras, se puede configurar la interfaz de usuario. Cree dos instancias de Button: una para enviar una palabra y otra para borrar una palabra que ya se ha formado. En cada caso, hay que responder a entradas de usuario detectando el evento `MouseEvent.CLICK` difundido por el botón y llamando a continuación a la función. En la función `setupUI()`, este código crea los detectores en los dos botones:

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

Generación de un tablero de juego

El tablero del juego es una cuadrícula de letras aleatorias. En la función `generateBoard()` se crea una cuadrícula bidimensional anidando un bucle dentro de otro. El primer bucle incrementa las filas y el segundo incrementa el número total de columnas por fila. Cada una de las celdas creadas por estas filas y columnas contiene un botón que representa una letra del tablero.

```
private function generateBoard(startX:Number, startY:Number, totalRows:Number,
totalCols:Number, buttonSize:Number):void
{
    buttons = new Array();
    var colCounter:uint;
    var rowCounter:uint;
    for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
    {
        for (colCounter = 0; colCounter < totalCols; colCounter++)
        {
            var b:Button = new Button();
            b.x = startX + (colCounter*buttonSize);
            b.y = startY + (rowCounter*buttonSize);
            b.addEventListener(MouseEvent.CLICK, letterClicked);
            b.label = getRandomLetter().toUpperCase();
            b.setSize(buttonSize,buttonSize);
            b.name = "buttonRow"+rowCounter+"Col"+colCounter;
            addChild(b);

            buttons.push(b);
        }
    }
}
```

Aunque sólo hay una línea que añade un detector para un evento `MouseEvent.CLICK`, como está en un bucle `for` se asignará un detector a cada instancia de Button. Además, a cada botón se le asigna un nombre derivado de su posición de fila y columna, lo que proporciona una manera sencilla de hacer referencia a la fila y columna de cada botón en otras partes del código.

Creación de palabras a partir de entradas de usuario

Las palabras pueden escribirse seleccionando letras contiguas horizontal o verticalmente, pero nunca se puede usar dos veces la misma letra. Cada clic genera un evento de ratón y hace que se compruebe que la palabra que el usuario está escribiendo es la continuación correcta de las letras en las que se hizo clic previamente. En caso contrario, se elimina la palabra anterior y se inicia otra nueva. Esta comprobación se produce en el método `isLegalContinuation()`.

```
private function isLegalContinuation(prevButton:Button, currButton:Button):Boolean
{
    var currButtonRow:Number = Number(currButton.name.charAt(currButton.name. indexOf("Row") +
3));
    var currButtonCol:Number = Number(currButton.name.charAt(currButton.name.indexOf("Col") +
3));
    var prevButtonRow:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Row") +
3));
    var prevButtonCol:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Col") +
3));

    return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow - currButtonRow) <= 1) ||
             (prevButtonRow == currButtonRow && Math.abs(prevButtonCol - currButtonCol) <= 1));
}
```

Los métodos `charAt()` e `indexOf()` de la clase String recuperan las filas y columnas adecuadas del botón en el que se acaba de hacer clic y del botón en el que se hizo clic previamente. El método `isLegalContinuation()` devuelve `true` si la fila o columna no cambia, y si la fila o columna que ha cambiado corresponde a un solo incremento con respecto a la anterior. Si se desea cambiar las reglas del juego y permitir la formación de palabras en diagonal, se pueden eliminar las comprobaciones de filas o columnas que no cambian; la línea final sería como la siguiente:

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol -
currButtonCol) <= 1));
```

Comprobación de las palabras formadas

Para completar el código del juego son necesarios mecanismos para comprobar las palabras formadas y actualizar la puntuación. El método `searchForWord()` realiza estas funciones:

```
private function searchForWord(str:String):Number
{
    if (words && str)
    {
        var i:uint = 0
        for (i = 0; i < words.length; i++)
        {
            var thisWord:String = words[i];
            if (str == words[i])
            {
                return i;
            }
        }
        return -1;
    }
    else
    {
        trace("WARNING: cannot find words, or string supplied is null");
    }
    return -1;
}
```

Esta función recorre todas las palabras del diccionario. Si la palabra del usuario coincide con una palabra del diccionario, se devuelve su posición en el diccionario. El método `submitWord()` comprueba entonces la respuesta y actualiza la puntuación si la posición es válida.

Personalización

Al principio de la clase hay varias constantes. Se puede modificar el juego modificando estas variables. Por ejemplo, se puede cambiar la cantidad de tiempo de juego incrementando la variable `TOTAL_TIME`. O se puede incrementar ligeramente la variable `PERCENT_VOWELS` para aumentar la probabilidad de encontrar palabras.

Capítulo 27: Redes y comunicación

En este capítulo se explica cómo activar el archivo SWF para comunicarse con archivos externos y otras instancias de Adobe Flash Player y Adobe AIR. También se explica cómo cargar datos de orígenes externos, enviar mensajes entre un servidor Java y Flash Player, y realizar cargas y descargas de archivos con las clases FileReference y FileReferenceList.

Fundamentos de redes y comunicación

Introducción a las redes y la comunicación

Cuando se generan aplicaciones de ActionScript más complejas, a menudo es necesario comunicarse con scripts de servidor o cargar datos de archivos de texto o XML externos. El paquete flash.net contiene clases para enviar y recibir datos a través de Internet; por ejemplo, para cargar contenido de URL remotos, comunicarse con otras instancias de Flash Player o AIR y conectarse a sitios Web remotos.

En ActionScript 3.0, se pueden cargar archivos externos con las clases URLLoader y URLRequest. Después se utiliza una clase específica para acceder a los datos, en función del tipo de datos que se haya cargado. Por ejemplo, si el contenido remoto tiene formato de pares nombre-valor, se utiliza la clase URLVariables para analizar los resultados del servidor. Como alternativa, si el archivo cargado con las clases URLLoader y URLRequest es un documento XML remoto, se puede analizar el documento XML con el constructor de la clase XML, el constructor de la clase XMLDocument o el método `XMLDocument.parseXML()`. Esto permite simplificar el código ActionScript porque el código que se emplea para cargar archivos externos es el mismo si se utilizan las clases XML o URLVariables, o cualquier otra clases para analizar y manipular los datos remotos.

El paquete flash.net también contiene clases para otros tipos de comunicación remota. Incluyen la clase FileReference para cargar y descargar archivos de un servidor, las clases Socket y XMLSocket que permiten la comunicación directa con equipos remotos a través de conexiones de socket, y las clases NetConnection y NetStream, que se utilizan para comunicarse con recursos de servidor específicos de Flash (como servidores de Flash Media Server y Flash Remoting) así como para cargar archivos de vídeo.

Por último, el paquete flash.net incluye clases para la comunicación en los equipos locales de los usuarios. Entre estas clases se incluyen la clase LocalConnection, que permite la comunicación entre dos o más archivos SWF que se ejecutan en un solo equipo, y la clase SharedObject, que permite almacenar datos en el equipo de un usuario y recuperarlos más adelante, cuando vuelven a la aplicación.

Tareas comunes de redes y comunicaciones

En la lista siguiente se describen las tareas más comunes relacionadas con la comunicación externa desde ActionScript, que se describirán en este capítulo:

- Cargar datos desde un archivo externo o script de servidor
- Enviar datos a un script de servidor
- Comunicarse con otros archivos SWF locales
- Trabajar con conexiones de socket binario
- Comunicarse con sockets XML

- Almacenar datos locales persistentes
- Cargar archivos en un servidor
- Descargar archivos desde un servidor al equipo del usuario

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Datos externos: datos que están almacenados en algún formato fuera del archivo SWF y se cargan en el archivo SWF cuando es necesario. Estos datos pueden estar almacenados en un archivo que se carga directamente o en una base de datos u otra forma, y se recuperan llamando a scripts o programas que se ejecutan en un servidor.
- Variables con codificación URL: el formato codificado en URL proporciona una manera de representar varias variables (pares de nombre de variable y valor) en una sola cadena de texto. Las variables individuales se escriben con el formato `nombre=valor`. Cada variable (es decir, cada par nombre-valor) se separa mediante caracteres ampersand, tal y como se indica: `variable1=value1&variable2=value2`. De esta manera se puede enviar un número arbitrario de variables en un solo mensaje.
- Tipo MIME: código estándar que se utiliza para identificar el tipo de un archivo determinado en la comunicación a través de Internet. Cualquier tipo de archivo tiene un código específico que se utiliza para su identificación. Cuando se envía un archivo o un mensaje, un programa (como un servidor Web o la instancia de Flash Player o AIR de un usuario) especifica el tipo de archivo que se envía.
- HTTP: protocolo de transferencia de hipertexto. Es un formato estándar para entregar páginas Web y otros tipos de contenido enviado a través de Internet.
- Método de solicitud: cuando un programa como Flash Player o un navegador Web envía un mensaje (denominado solicitud HTTP) a un servidor Web, los datos enviados pueden incorporarse en la solicitud de dos maneras distintas: los dos *métodos de solicitud* GET y POST. En el extremo del servidor, el programa que recibe la solicitud tendrá que mirar en la parte apropiada de la solicitud para encontrar los datos, por lo que el método de solicitud utilizado para enviar datos desde ActionScript debe coincidir con el método de solicitud utilizado para leer los datos en el servidor.
- Conexión de socket: conexión persistente para la comunicación entre dos equipos.
- Carga: envío de un archivo a otro equipo.
- Descarga: recuperación de un archivo de otro equipo.

Trabajo con direcciones IPv6

Flash Player 9.0.115.0 y las versiones posteriores admiten IPv6 (Protocolo de Internet versión 6). IPv6 es una versión del Protocolo de Internet que admite direcciones de 128 bits (una mejora con respecto al protocolo anterior IPv4 que admite direcciones de 32 bits). Es posible que tenga que activar IPv6 en sus interfaces de red. Para más información, consulte la Ayuda del sistema operativo que aloja los datos.

Si se admite IPv6 en el sistema de host, puede especificar direcciones literales numéricas de IPv6 en URL entre corchetes ([]), como se muestra a continuación:

```
rtmp://[2001:db8:ccc3:ffff:0:444d:555e:666f]:1935/test
```

Flash Player devuelve valores IPv6 literales, en función de las siguiente reglas:

- Flash Player devuelve el formulario largo de la cadena para las direcciones IPv6.
- El valor IP no tiene abreviaciones de dos puntos dobles.
- Los dígitos hexadecimales sólo aparecen en minúscula.

- Las direcciones IPv6 aparecen entre corchetes ([]).
- Cada cuarteto de direcciones se representa como dígitos hexadecimales de 0 a 4, donde se omiten los primeros ceros.
- Un cuarteto de direcciones de todos los ceros aparece como un solo cero (no dos puntos dobles), excepto donde se indica en la siguiente lista de excepciones.

Los valores IPv6 que devuelve Flash Player presentan las siguientes excepciones:

- Una dirección IPv6 no especificada (todos los ceros) aparece como [::].
- La dirección IPv6 de localhost o bucle invertido se presenta como [::1].
- Las direcciones asignadas a IPv4 (convertidas a IPv6) se presentan como [::ffff:a.b.c.d], siendo a.b.c.d un valor típico decimal con puntos de IPv4.
- Las direcciones compatibles con IPv4 aparecen como [::a.b.c.d], siendo a.b.c.d un valor típico decimal con puntos de IPv4.

Ejecución de los ejemplos del capítulo

A medida que progrese en el estudio de este capítulo, podría desear probar los listados de código de ejemplo. Varios de los listados de código de este capítulo cargan datos externos o realizan algún otro tipo de comunicación; a menudo estos ejemplos incluyen llamadas a la función `trace()` que muestran los resultados de la ejecución del ejemplo en el panel Salida. Otros ejemplos realizan alguna función, como cargar un archivo a un servidor. Para probar esos ejemplos hay que interactuar con el archivo SWF y confirmar que realizan la acción que dicen realizar.

Los ejemplos de código se clasifican en dos categorías. Algunos de los listados de ejemplo se han escrito suponiendo que el código está en un script autónomo (por ejemplo, asociado a un fotograma clave en un documento Flash). Para probar estos ejemplos:

1. Cree un documento de Flash nuevo.
2. Seleccione el fotograma clave del Fotograma 1 de la línea de tiempo y abra el panel Acciones.
3. Copie el listado de código en el panel Script.
4. En el menú principal, elija Control > Probar película para crear el archivo SWF y probar el ejemplo.

Otros listados de código de ejemplo están escritos como una clase, con el fin de que la clase de ejemplo sirva como clase de documento para el documento de Flash. Para probar estos ejemplos:

1. Cree un documento de Flash vacío y guárdelo en el equipo.
2. Cree un nuevo archivo de ActionScript y guárdelo en el mismo directorio que el documento de Flash. El nombre del archivo debe coincidir con el nombre de la clase del listado de código. Por ejemplo, si el listado de código define una clase denominada "UploadTest", guarde el archivo ActionScript como "UploadTest.as".
3. Copie el listado de código en el archivo de ActionScript y guarde el archivo.
4. En el documento de Flash, haga clic en una parte vacía del escenario o el espacio de trabajo para activar el inspector de propiedades del documento.
5. En el inspector de propiedades, en el campo Clase de documento, escriba el nombre de la clase de ActionScript que copió del texto.
6. Ejecute el programa seleccionando Control > Probar película y pruebe el ejemplo.

Por último, algunos de los ejemplos del capítulo implican interactuar con un programa en ejecución en un servidor. Estos ejemplos incluyen código que se pueden utilizar para crear el programa de servidor necesario para probarlos; será necesario configurar las aplicaciones apropiadas en un equipo servidor Web para probar estos ejemplos.

Trabajo con datos externos

ActionScript 3.0 incluye mecanismos para cargar datos desde fuentes externas. Estas fuentes pueden ser de contenido estático, como archivos de texto, o de contenido dinámico, como un script Web que recupera datos de una base de datos. Se puede aplicar formato a los datos de varias maneras y ActionScript proporciona funcionalidad para descodificar y acceder a los datos. También se pueden enviar datos al servidor externo como parte del proceso de recuperación de datos.

Utilización de las clases URLLoader y URLVariables

ActionScript 3.0 utiliza las clases URLLoader y URLVariables para cargar datos externos. La clase URLLoader descarga datos desde una URL como texto, datos binarios o variables con codificación URL. La clase URLLoader es útil para descargar archivos de texto, XML u otra información para utilizarla en aplicaciones ActionScript dinámicas basadas en datos. La clase URLLoader aprovecha el modelo avanzado de gestión de eventos de ActionScript 3.0, que permite detectar eventos como `complete`, `httpStatus`, `ioError`, `open`, `progress` y `securityError`. El nuevo modelo de gestión de eventos supone una mejora significativa con respecto al uso de los controladores de eventos `LoadVars.onData`, `LoadVars.onHTTPStatus` y `LoadVars.onLoad` en ActionScript 2.0, ya que permite gestionar errores y eventos con mayor eficacia. Para obtener más información sobre la gestión de eventos, consulte "Gestión de eventos" en la página 254.

De forma muy similar a las clases XML y LoadVars de versiones anteriores de ActionScript, los datos del URL URLLoader no están disponibles hasta que finaliza la descarga. Para supervisar el progreso de la descarga (bytes cargados y bytes totales) es necesario detectar el evento `flash.events.ProgressEvent.PROGRESS` que se va a distribuir, aunque si un archivo se carga demasiado rápido, puede ser que no se distribuya un evento `ProgressEvent.PROGRESS`. Cuando finaliza la descarga de un archivo, se distribuye el evento `flash.events.Event.COMPLETE`. Los datos cargados se descodifican de la codificación UTF-8 o UTF-16 y se convierten en una cadena.

Nota: *si no se establece ningún valor para* `URLRequest.contentType`*, los valores se envían como* `application/x-www-form-urlencoded`*.*

El método `URLLoader.load()` (y opcionalmente el constructor de la clase URLLoader) admite un solo parámetro, `request`, que es una instancia de URLRequest. Una instancia de URLRequest contiene toda la información de una sola petición HTTP, como la URL de destino, el método de petición (`GET` o `POST`), información de encabezado adicional y el tipo MIME (por ejemplo, cuando se carga contenido XML).

Por ejemplo, para cargar un paquete XML en un script de servidor, se podría usar el siguiente código ActionScript 3.0:

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
    <login>
        <time>{secondsUTC}</time>
        <username>Ernie</username>
        <password>guru</password>
    </login>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/login.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
try
{
    loader.load(request);
}
catch (error:ArgumentError)
{
    trace("An ArgumentError has occurred.");
}
catch (error:SecurityError)
{
    trace("A SecurityError has occurred.");
}
```

El fragmento de código anterior crea una instancia de XML denominada `dataXML` que contiene un paquete XML que se enviará al servidor. A continuación, se establece la propiedad `contentType` de URLRequest en `"text/xml"` y la propiedad `data` de URLRequest en el contenido del paquete XML, que se convierte en una cadena mediante el método `XML.toXMLString()`. Finalmente, se crea una nueva instancia de URLLoader y se envía la petición al script remoto mediante el método `URLLoader.load()`.

Hay tres formas posibles de especificar parámetros para pasarlos en una petición de URL:

- En el constructor de URLVariables
- En el método `URLVariables.decode()`
- Como propiedades específicas en el propio objeto URLVariables

Cuando se definen variables en el constructor de URLVariables o en el método `URLVariables.decode()`, hay que asegurarse de que se codifica como URL el carácter ampersand porque tiene un significado especial y actúa como delimitador. Por ejemplo, si se pasa un ampersand, es necesario codificarlo como URL cambiando `&` por `%26`, ya que el ampersand actúa como delimitador de los parámetros.

Carga de datos desde documentos externos

Cuando se generan aplicaciones dinámicas con ActionScript 3.0, es recomendable cargar datos de archivos externos o scripts de servidor. De esta forma es posible generar aplicaciones dinámicas sin tener que editar o recompilar los archivos de ActionScript. Por ejemplo, si se genera una aplicación de "sugerencia del día", se puede escribir un script de servidor que recupere una sugerencia aleatoria de una base de datos y la guarde en un archivo de texto una vez al día. Luego la aplicación ActionScript puede cargar el contenido de un archivo de texto estático en lugar de consultar la base de datos cada vez.

El siguiente fragmento de código crea un objeto URLRequest y URLLoader, que carga el contenido de un archivo de texto externo, params.txt:

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

Se puede simplificar el fragmento de código anterior del siguiente modo:

```
var loader:URLLoader = new URLLoader(new URLRequest("params.txt"));
```

De forma predeterminada, si no se define un método de petición, Flash Player y Adobe AIR cargan el contenido con el método HTTP `GET`. Si se desea enviar los datos con el método `POST`, es necesario establecer la propiedad `request.method` en `POST` con la constante estática `URLRequestMethod.POST`, como se muestra en el siguiente código:

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

El documento externo, params.txt, que se carga en tiempo de ejecución contiene los siguientes datos:

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,Dece
mber&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

El archivo contiene dos parámetros: `monthNames` y `dayNames`. Cada parámetro contiene una lista separada por comas que se analiza como cadenas. Esta lista se puede dividir en un conjunto, mediante el método `String.split()`.

Conviene evitar la utilización de palabras reservadas y construcciones del lenguaje como nombres de variables en archivos de datos externos, ya que dificulta la lectura y depuración del código.

Cuando se han cargado los datos, se distribuye el evento `Event.COMPLETE` y el contenido del documento externo está disponible para utilizarlo en la propiedad `data` de URLLoader, como se muestra en el siguiente código:

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    trace(loader2.data);
}
```

Si el documento remoto contiene pares nombre-valor, se pueden analizar los datos con la clase URLVariables, si se pasa el contenido del archivo cargado, como se muestra a continuación:

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

Cada par nombre-valor del archivo externo se crea como una propiedad del objeto URLVariables. Cada propiedad del objeto de variables del ejemplo de código anterior se trata como una cadena. Si el valor del par nombre-valor es una lista de elementos, se puede convertir la cadena en un conjunto, mediante una llamada al método `String.split()`, como se muestra a continuación:

```
var dayNameArray:Array = variables.dayNames.split(",");
```

Si se cargan datos numéricos de archivos de texto externos, es necesario convertir los valores en valores numéricos, mediante una función de nivel superior como `int()`, `uint()` o `Number()`.

En lugar de cargar el contenido del archivo remoto como una cadena y crear un nuevo objeto URLVariables, se puede establecer la propiedad `URLLoader.dataFormat` en una de las propiedades estáticas de la clase URLLoaderDataFormat. Los tres valores posibles para la propiedad `URLLoader.dataFormat` son los siguientes:

- `URLLoaderDataFormat.BINARY`: la propiedad `URLLoader.data` contendrá datos binarios almacenados en un objeto ByteArray.
- `URLLoaderDataFormat.TEXT`: la propiedad `URLLoader.data` contendrá texto en un objeto String.
- `URLLoaderDataFormat.VARIABLES`: la propiedad `URLLoader.data` contendrá variables con codificación URL almacenadas en un objeto URLVariables.

En el código siguiente se muestra que, al establecer la propiedad `URLLoader.dataFormat` en `URLLoaderDataFormat.VARIABLES`, se permite analizar automáticamente los datos cargados en un objeto URLVariables:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class URLLoaderDataFormatExample extends Sprite
    {
        public function URLLoaderDataFormatExample()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/params.txt");
            var variables:URLLoader = new URLLoader();
            variables.dataFormat = URLLoaderDataFormat.VARIABLES;
            variables.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                variables.load(request);
            }
            catch (error:Error)
            {
                trace("Unable to load URL: " + error);
            }
        }
        private function completeHandler(event:Event):void
        {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
        }
    }
}
```

***Nota:** el valor predeterminado de `URLLoader.dataFormat` es `URLLoaderDataFormat.TEXT`.*

Como se muestra en el siguiente ejemplo, cargar XML de un archivo externo equivale a cargar URLVariables. Se puede crear una instancia de URLRequest y una instancia de URLLoader, y utilizarlas para descargar un documento XML remoto. Cuando el archivo se ha descargado completamente, se distribuye el evento `Event.COMPLETE` y el contenido del archivo externo se convierte en una instancia de XML, que puede analizarse con los métodos y propiedades de XML.

```
package
{
    import flash.display.Sprite;
    import flash.errors.*;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class ExternalDocs extends Sprite
    {
        public function ExternalDocs()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/data.xml");
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                loader.load(request);
            }
            catch (error:ArgumentError)
            {
                trace("An ArgumentError has occurred.");
            }
            catch (error:SecurityError)
            {
                trace("A SecurityError has occurred.");
            }
        }
        private function completeHandler(event:Event):void
        {
            var dataXML:XML = XML(event.target.data);
            trace(dataXML.toXMLString());
        }
    }
}
```

Comunicación con scripts externos

Además de cargar archivos de datos externos, se puede utilizar la clase URLVariables para enviar variables a un script de servidor y procesar la respuesta del servidor. Esto resulta útil, por ejemplo, si se programa un juego y se desea enviar la puntuación del usuario a un servidor para calcular si debe añadirse o no a la lista de mejores puntuaciones, o incluso enviar la información de inicio de sesión de un usuario a un servidor para que lo valide. Un script de servidor puede procesar el nombre de usuario y la contraseña, validarla en una base de datos y devolver la confirmación de si las credenciales suministradas del usuario son válidas.

El siguiente fragmento de código crea un objeto URLVariables denominado `variables`, que crea una nueva variable denominada `name`. A continuación, se crea un objeto URLRequest que especifica el URL del script de servidor donde se enviarán las variables. Luego se establece la propiedad `method` del objeto URLRequest para enviar las variables como una petición HTTP `POST`. Para añadir el objeto URLVariables a la petición de URL, se establece la propiedad `data` del objeto URLRequest en el objeto URLVariables creado previamente. Finalmente, se crea la instancia de URLLoader y se llama al método `URLLoader.load()`, que inicia la petición.

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
    loader.load(request);
}
catch (error:Error)
{
    trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
    trace(event.target.data.welcomeMessage);
}
```

El código siguiente incluye el contenido del documento greeting.cfm de Adobe ColdFusion® utilizado en el ejemplo anterior:

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

Conexión con otras instancias de Flash Player y AIR

La clase LocalConnection permite la comunicación entre distintas instancias de Flash Player y AIR como, por ejemplo, un archivo SWF en un contenedor HTML o en un reproductor autónomo o incorporado. Esto permite generar aplicaciones muy versátiles que pueden compartir datos entre instancias de Flash Player y AIR como, por ejemplo, archivos SWF ejecutados en un navegador Web o incorporados en aplicaciones de escritorio.

Clase LocalConnection

La clase LocalConnection permite desarrollar archivos SWF capaces de enviar instrucciones a otros archivos SWF sin utilizar el método `fscommand()` ni JavaScript. Los objetos LocalConnection sólo pueden comunicarse con archivos SWF que se ejecuten en el mismo equipo cliente, aunque pueden ejecutarse en aplicaciones distintas. Por ejemplo, un archivo SWF que se ejecuta en un navegador y un archivo SWF que se ejecuta en un proyector pueden compartir información, y el proyector conserva la información local y el archivo basado en el navegador se conecta de forma remota. (Un proyector es un archivo SWF guardado en un formato que puede ejecutarse como una aplicación autónoma, es decir, que el proyector no requiere que se instale Flash Player porque está incorporado en el ejecutable.)

Los objetos LocalConnection se pueden utilizar para comunicarse entre archivos SWF con distintas versiones de ActionScript:

- Los objetos LocalConnection de ActionScript 3.0 pueden comunicarse con objetos LocalConnection creados con ActionScript 1.0 ó 2.0.

- Los objetos LocalConnection de ActionScript 1.0 ó 2.0 pueden comunicarse con objetos LocalConnection creados con ActionScript 3.0.

Flash Player controla automáticamente esta comunicación entre objetos LocalConnection de distintas versiones.

La manera más sencilla de utilizar un objeto LocalConnection es permitir la comunicación únicamente entre objetos LocalConnection del mismo dominio. De esta manera no hay que preocuparse por la seguridad. No obstante, si necesita permitir la comunicación entre dominios, existen diversas formas de implementar medidas de seguridad. Para obtener más información, consulte la explicación del parámetro `connectionName` del método`send()` y las entradas `allowDomain()` y `domain` en el listado de clases LocalConnection en la Referencia del lenguaje y componentes ActionScript 3.0.


Es posible utilizar objetos LocalConnection para enviar y recibir datos en un solo archivo SWF, pero Adobe no recomienda esta práctica. En lugar de eso, se recomienda utilizar objetos compartidos.

Hay tres formas de añadir métodos callback a los objetos LocalConnection:

- Crear una subclase de la clase LocalConnection y añadir métodos.
- Establecer la propiedad `LocalConnection.client` en un objeto que implemente los métodos.
- Crear una clase dinámica que amplíe LocalConnection y asociar métodos de forma dinámica.

La primera forma de añadir métodos callback consiste en ampliar la clase LocalConnection. Los métodos se definen en la clase personalizada en lugar de añadirlos de forma dinámica a la instancia de LocalConnection. Este enfoque se muestra en el siguiente código:

```
package
{
    import flash.net.LocalConnection;
    public class CustomLocalConnection extends LocalConnection
    {
        public function CustomLocalConnection(connectionName:String)
        {
            try
            {
                connect(connectionName);
            }
            catch (error:ArgumentError)
            {
                // server already created/connected
            }
        }
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

Para crear una nueva instancia de la clase DynamicLocalConnection, se puede usar el siguiente código:

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

La segunda forma de añadir métodos callback consiste en utilizar la propiedad `LocalConnection.client`. Esto implica crear una clase personalizada y asignar una nueva instancia a la propiedad `client`, como se muestra en el siguiente código:

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

La propiedad `LocalConnection.client` indica los métodos callback del objeto que deben llamarse. En el código anterior, la propiedad `client` se estableció en una nueva instancia de una clase personalizada, CustomClient. El valor predeterminado de la propiedad `client` es la instancia de LocalConnection actual. Se puede utilizar la propiedad `client` si se tienen dos controladores de datos con el mismo conjunto de métodos pero distintos comportamientos; por ejemplo, en una aplicación donde un botón de una ventana activa o desactiva la visualización en una segunda ventana.

Para crear la clase CustomClient, se podría usar el siguiente código:

```
package
{
    public class CustomClient extends Object
    {
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

La tercera forma de añadir métodos callback, que consiste en crear una clase dinámica y asociar los métodos de forma dinámica, es muy similar a la utilización de la clase LocalConnection en versiones anteriores de ActionScript, como se muestra en el siguiente código:

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

Los métodos callback pueden añadirse de forma dinámica a esta clase con el siguiente código:

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Add your code here.
public function onMethod(timeString:String):void
{
    trace("onMethod called at: " + timeString);
}
```

No se recomienda la anterior forma de añadir métodos callback porque el código tiene una escasa portabilidad. Además, si se utiliza este método de creación de conexiones locales, podrían surgir problemas de rendimiento debido a que el acceso a las propiedades dinámicas es considerablemente más lento que el acceso a las propiedades cerradas.

Envío de mensajes entre dos instancias de Flash Player

La clase LocalConnection se utiliza en la comunicación entre distintas instancias de Flash Player y Adobe AIR. Por ejemplo, se pueden tener varias instancias de Flash Player en una página Web o se puede hacer que una instancia de Flash Player recupere datos de otra instancia de Flash Player en una ventana emergente.

El código siguiente define un objeto de conexión local que actúa como un servidor y acepta llamadas entrantes de otras instancias de Flash Player:

```
package
{
    import flash.net.LocalConnection;
    import flash.display.Sprite;
    public class ServerLC extends Sprite
    {
        public function ServerLC()
        {
            var lc:LocalConnection = new LocalConnection();
            lc.client = new CustomClient1();
            try
            {
                lc.connect("conn1");
            }
            catch (error:Error)
            {
                trace("error:: already connected");
            }
        }
    }
}
```

Este código primero crea un objeto LocalConnection denominado `lc` y establece la propiedad `client` en una clase personalizada, CustomClient1. Cuando otra instancia de Flash Player llama a un método en esta instancia de conexión local, Flash Player busca dicho método en la clase CustomClient1.

Cuando una instancia de Flash Player se conecta a este archivo SWF e intenta llamar a cualquier método en la conexión local especificada, la petición se envía a la clase especificada por la propiedad `client`, que se establece en la clase CustomClient1:

```
package
{
    import flash.events.*;
    import flash.system.fscommand;
    import flash.utils.Timer;
    public class CustomClient1 extends Object
    {
        public function doMessage(value:String = ""):void
        {
            trace(value);
        }
        public function doQuit():void
        {
            trace("quitting in 5 seconds");
            this.close();
            var quitTimer:Timer = new Timer(5000, 1);
            quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
        }
        public function closeHandler(event:TimerEvent):void
        {
            fscommand("quit");
        }
    }
}
```

Para crear un servidor LocalConnection, hay que llamar al método `LocalConnection.connect()` y proporcionar un nombre de conexión exclusivo. Si ya se dispone de una conexión con el nombre especificado, se genera un error ArgumentError, que indica que el intento de conexión falló porque el objeto ya está conectado.

El siguiente fragmento de código muestra la creación de una nueva conexión de socket con el nombre `conn1`:

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error! Server already exists\n");
}
```

La conexión al archivo SWF principal desde un archivo SWF secundario requiere crear un nuevo objeto LocalConnection en el objeto LocalConnection emisor y luego llamar al método `LocalConnection.send()` con el nombre de la conexión y el nombre del método que se va a ejecutar. Por ejemplo, para conectarse al objeto LocalConnection creado anteriormente, se utiliza el siguiente código:

```
sendingConnection.send("conn1", "doQuit");
```

Este código se conecta a un objeto LocalConnection existente, con el nombre de conexión `conn1`, y llama al método `doQuit()` en el archivo SWF remoto. Si se desea enviar parámetros al archivo SWF remoto, se deben especificar argumentos adicionales después del nombre del método en el método `send()`, como se muestra en el siguiente fragmento de código:

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

Conexión a documentos SWF de distintos dominios

Para permitir las comunicaciones exclusivamente desde dominios específicos, hay que llamar al método `allowDomain()` o `allowInsecureDomain()` de la clase LocalConnection y pasar una lista de uno o varios dominios que tienen permitido el acceso a este objeto LocalConnection.

En versiones anteriores de ActionScript, `LocalConnection.allowDomain()` y `LocalConnection.allowInsecureDomain()` eran métodos callback que implementaban los desarrolladores y que tenían que devolver un valor booleano. En ActionScript 3.0, `LocalConnection.allowDomain()` y `LocalConnection.allowInsecureDomain()` son métodos incorporados a los que los desarrolladores llaman del mismo modo que `Security.allowDomain()` y `Security.allowInsecureDomain()`, pasando uno o varios nombres de dominios que deben autorizarse.

Existen dos valores especiales que se pueden transmitir a los métodos `LocalConnection.allowDomain()` y `LocalConnection.allowInsecureDomain()`: `*` y `localhost`. El valor de asterisco `(*)` permite el acceso desde todos los dominios. La cadena `localhost` permite las llamadas al archivo SWF desde archivos SWF instalados localmente.

Flash Player 8 introdujo restricciones de seguridad en los archivos SWF locales. Un archivo SWF al que se le permite acceder a Internet no puede tener también acceso al sistema de archivos local. Si se especifica `localhost`, cualquier archivo SWF local podrá acceder al archivo SWF. Si el método `LocalConnection.send()` intenta comunicarse con un archivo SWF desde un entorno limitado de seguridad al que no puede acceder el código que realiza la llamada, se distribuye un evento `securityError(SecurityErrorEvent.SECURITY_ERROR)`. Para solucionar este error, se puede especificar el dominio del que realiza la llamada en el método `LocalConnection.allowDomain()`.

Si se implementa la comunicación exclusivamente entre archivos SWF del mismo dominio, se puede especificar un parámetro `connectionName` que no empiece por un carácter de subrayado (_) y que no especifique un nombre de dominio (por ejemplo, `myDomain:connectionName`). Debe utilizarse la misma cadena en el comando `LocalConnection.connect(connectionName)`.

Si se implementa la comunicación entre archivos SWF de distintos dominios, se debe especificar un parámetro `connectionName` cuyo valor empiece por un carácter de subrayado. La especificación del carácter de subrayado aumenta la portabilidad entre dominios del archivo SWF con el objeto LocalConnection receptor. Estos son los dos casos posibles:

- Si la cadena para `connectionName` no empieza por un carácter de subrayado, Flash Player añade un prefijo con el nombre de superdominio y dos puntos (por ejemplo, `myDomain:connectionName`). Aunque esto garantiza que la conexión no entre en conflicto con conexiones de otros dominios que tengan el mismo nombre, los objetos LocalConnection emisores deben especificar este superdominio (por ejemplo, `myDomain:connectionName`). Si el archivo SWF que contiene el objeto LocalConnection receptor se traslada a otro dominio, Flash Player cambiará el prefijo para reflejar el nuevo superdominio (por ejemplo, `anotherDomain:connectionName`). Sería necesario editar manualmente todos los objetos LocalConnection emisores para que señalaran al nuevo superdominio.
- Si la cadena de `connectionName` empieza por un carácter de subrayado (por ejemplo, `_connectionName`), Flash Player no añadirá ningún prefijo a la cadena. Esto significa que los objetos LocalConnection receptores y emisores utilizarán cadenas idénticas para `connectionName`. Si el objeto receptor utiliza `LocalConnection.allowDomain()` para especificar que se acepten las conexiones de cualquier dominio, el SWF que contiene el objeto LocalConnection receptor podrá trasladarse a otro dominio sin modificar ningún objeto LocalConnection emisor.

Conexiones de socket

Existen dos tipos diferentes de conexiones de socket posibles en ActionScript 3.0: conexiones de socket XML y conexiones de socket binario. Un socket XML permite conectarse a un servidor remoto y crear una conexión de servidor que permanece abierta hasta que se cierra de forma explícita. Esto permite intercambiar datos XML entre un servidor y un cliente sin tener que abrir continuamente nuevas conexiones de servidor. Otra ventaja de utilizar un servidor de socket XML es que el usuario no tiene que solicitar datos de forma explícita. Se pueden enviar datos desde el servidor, sin necesidad de peticiones, a cada cliente conectado al servidor de socket XML.

Las conexiones de socket XML requieren la presencia de un archivo de política de sockets en el servidor de destino. Para obtener más información, consulte "Controles de sitio Web (archivos de política)" en la página 721 y "Conexión a sockets" en la página 737.

Una conexión de socket binario es similar a un socket XML, pero el cliente y el servidor no necesitan intercambiar paquetes XML específicamente. En lugar de eso, la conexión puede transferir datos como información binaria. Esto permite conectar una amplia gama de servicios, como servidores de correo (POP3, SMTP e IMAP) y servidores de noticias (NNTP).

Clase Socket

La clase Socket, introducida en ActionScript 3.0, permite que el código ActionScript realice conexiones de socket, y que lea y escriba datos binarios sin formato. Es similar a la clase XMLSocket, pero no dicta el formato de los datos recibidos y transmitidos. La clase Socket resulta útil para interoperar con servidores que utilicen protocolos binarios. Se pueden utilizar conexiones de socket binario para escribir código que permita la interacción con varios protocolos de Internet distintos, como POP3, SMTP, IMAP y NNTP. Esto, a su vez, permite a Flash Player conectarse a servidores de correos y noticias.

Flash Player puede interactuar con un servidor directamente mediante el protocolo binario de dicho servidor. Algunos servidores utilizan el orden de bytes bigEndian y otros utilizan el orden de bytes littleEndian. La mayoría de los servidores de Internet utilizan el orden de bytes bigEndian, ya que el "orden de bytes de la red" es bigEndian. El orden de bytes littleEndian es popular porque la arquitectura Intel® x86 lo utiliza. Debe utilizarse el orden de bytes Endian que coincida con el orden de bytes del servidor que envía o recibe los datos. Todas las operaciones que se realizan mediante las interfaces IDataInput e IDataOutput, y las clases que implementan dichas interfaces (ByteArray, Socket y URLStream) se codifican de forma predeterminada en formato bigEndian. Esto supone que el byte más significativo va primero. El objetivo es que coincida el orden de bytes de la red oficial con el de Java. Para cambiar entre el uso de orden de bytes bigEndian o littleEndian, se puede establecer la propiedad `endian` en `Endian.BIG_ENDIAN` o `Endian.LITTLE_ENDIAN`.

La clase Socket hereda todos los métodos implementados por las interfaces IDataInput e IDataOutput (ubicadas en el paquete flash.utils) y dichos métodos deben utilizarse para escribir y leer en Socket.

Clase XMLSocket

ActionScript proporciona una clase XMLSocket incorporada que permite abrir una conexión continua con un servidor. Esta conexión abierta elimina los problemas de latencia y se usa con frecuencia para las aplicaciones en tiempo real como las aplicaciones de chat o los juegos de varios jugadores. Una solución de chat tradicional basada en HTTP suele sondear el servidor y descargar los mensajes nuevos mediante una petición HTTP. Por contra, una solución de chat con XMLSocket mantiene una conexión abierta con el servidor, lo que permite que el servidor envíe de inmediato los mensajes entrantes sin que se produzca una petición del cliente.

Para crear una conexión de socket, se debe crear una aplicación de servidor que espere la petición de conexión de socket y envíe una respuesta al archivo SWF. Este tipo de aplicación de servidor puede escribirse en un lenguaje de programación como Java, Python o Perl. Para utilizar la clase XMLSocket, el equipo servidor debe ejecutar un demonio que entienda el protocolo utilizado por la clase XMLSocket. El protocolo se describe en la siguiente lista:

- Los mensajes XML se envían a través de una conexión de socket ininterrumpida TCP/IP dúplex.
- Cada mensajes XML es un documento XML completo terminado en un byte cero (0).
- Pueden enviarse y recibirse un número ilimitado de mensajes XML a través de una misma conexión XMLSocket.

La clase XMLSocket no puede atravesar cortafuegos automáticamente, ya que, a diferencia del protocolo RTMP (Real-Time Messaging Protocol), XMLSocket no dispone de prestaciones de tunelación HTTP. Si es necesario utilizar tunelación HTTP, debe considerarse la posibilidad de utilizar Flash Remoting o Flash Media Server (que admite RTMP).

***Nota:** la configuración de un servidor para que se comunique con el objeto XMLSocket puede resultar compleja. Si la aplicación no requiere interactividad en tiempo real, se debe usar la clase URLLoader en vez de la clase XMLSocket.*

Se pueden utilizar los métodos `XMLSocket.connect()` y `XMLSocket.send()` de la clase XMLSocket para transferir XML desde y hacia un servidor, a través de una conexión de socket. El método `XMLSocket.connect()` establece una conexión de socket con un puerto de servidor Web. El método `XMLSocket.send()` pasa un objeto XML al servidor especificado en la conexión de socket.

Cuando se llama al método `XMLSocket.connect()`, Flash Player abre una conexión TCP/IP con el servidor y la mantiene abierta hasta que se produce una de las siguientes situaciones:

- Se llama al método `XMLSocket.close()` de la clase XMLSocket.
- No existen más referencias al objeto XMLSocket.
- Se sale de Flash Player.
- Se interrumpe la conexión (por ejemplo, se desconecta el módem).

Creación y conexión de un servidor de socket XML de Java

El siguiente código muestra un sencillo servidor XMLSocket escrito en Java que acepta conexiones entrantes y muestra los mensajes recibidos en la ventana de símbolo del sistema. De forma predeterminada, se crea un nuevo servidor en el puerto 8080 del equipo local, aunque se puede especificar otro número de puerto cuando se inicia el servidor desde la línea de comandos.

Cree un nuevo documento de texto y añada el siguiente código:

```
import java.io.*;
import java.net.*;

class SimpleServer
{
    private static SimpleServer server;
    ServerSocket socket;
    Socket incoming;
    BufferedReader readerIn;
    PrintStream printOut;

    public static void main(String[] args)
    {
        int port = 8080;

        try
        {
            port = Integer.parseInt(args[0]);
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            // Catch exception and keep going.
        }

        server = new SimpleServer(port);
    }

    private SimpleServer(int port)
    {
        System.out.println(">> Starting SimpleServer");
        try
        {
            socket = new ServerSocket(port);
            incoming = socket.accept();
            readerIn = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
            printOut = new PrintStream(incoming.getOutputStream());
            printOut.println("Enter EXIT to exit.\r");
            out("Enter EXIT to exit.\r");
            boolean done = false;
            while (!done)
            {
                String str = readerIn.readLine();
                if (str == null)
                {
                    done = true;
                }
                else
                {
```

```
                    out("Echo: " + str + "\r");
                    if(str.trim().equals("EXIT"))
                    {
                        done = true;
                    }
                }
                incoming.close();
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }

    private void out(String str)
    {
        printOut.println(str);
        System.out.println(str);
    }
}
```

Guarde el documento en el disco duro como SimpleServer.java y compílelo con la ayuda de un compilador de Java, que crea un archivo de clase de Java denominado SimpleServer.class.

Para iniciar el servidor XMLSocket, abra un símbolo del sistema y escriba `java SimpleServer`. El archivo SimpleServer.class puede almacenarse en cualquier lugar del equipo local o la red; no es necesario que esté en el directorio raíz del servidor Web.

Si no puede iniciar el servidor porque los archivos no se encuentran en la ruta de clases de Java, intente iniciar el servidor con `java -classpath. SimpleServer`.

Para conectarse a XMLSocket desde la aplicación ActionScript, es necesario crear una nueva instancia de la clase XMLSocket y llamar al método `XMLSocket.connect()`, pasando un nombre de host y un número de puerto, del siguiente modo:

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

Cuando se reciben datos del servidor, se distribuye el evento `data` (`flash.events.DataEvent.DATA`):

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + event.data);
}
```

Para enviar datos al servidor XMLSocket, se utiliza el método `XMLSocket.send()` y se pasa una cadena u objeto XML. Flash Player convierte el parámetro suministrado en un objeto String y envía el contenido al servidor XMLSocket, seguido de un byte cero (0):

```
xmlsock.send(xmlFormattedData);
```

El método `XMLSocket.send()` no devuelve ningún valor que indique si los datos se han transmitido correctamente. Si se produce un error al intentar enviar datos, se emite un error IOError.

Cada mensaje que se envía al servidor de socket XML debe terminar con un carácter de nueva línea (`\n`).

Almacenamiento de datos locales

Un objeto compartido, a menudo referido como "cookie de Flash", es un archivo de datos que se puede crear en el equipo a partir de los sitios que se visitan. Los objetos compartidos se utilizan sobre todo para mejorar la navegación en Web ya que, por ejemplo, permiten personalizar el aspecto de un sitio Web que se visita con frecuencia. Los objetos compartidos, por sí mismos, no actúan de ningún modo con los datos del equipo. Y lo que es más importante, los objetos compartidos nunca pueden acceder a direcciones de correo electrónico u otra información personal, ni recordar estos datos, a menos que el usuario proporcione conscientemente dicha información.

Para crear nuevas instancias de objetos compartidos, se utilizan los métodos estáticos `SharedObject.getLocal()` o `SharedObject.getRemote()`. El método `getLocal()` intenta cargar localmente un objeto compartido persistente que sólo está disponible para el cliente actual, mientras que el método `getRemote()` intenta cargar un objeto compartido remoto que puede compartirse entre varios clientes a través de un servidor como Flash Media Server. Si el objeto compartido local o remoto no existe, los métodos `getLocal()` y `getRemote()` crearán una nueva instancia de SharedObject.

El código siguiente intenta cargar un objeto compartido local denominado `test`. Si este objeto compartido no existe, se creará un nuevo objeto compartido con este nombre.

```
var so:SharedObject = SharedObject.getLocal("test");
trace("SharedObject is " + so.size + " bytes");
```

Si no se encuentra un objeto compartido denominado test, se crea uno nuevo con un tamaño de 0 bytes. Si el objeto compartido ya existía previamente, se devuelve su tamaño real, en bytes.

Para almacenar datos en un objeto compartido, hay que asignar valores al objeto de datos, como se muestra en el siguiente ejemplo:

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.now = new Date().time;
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

Si ya hay un objeto compartido con el nombre `test` y el parámetro `now`, se sobrescribe el valor existente. Se puede utilizar la propiedad `SharedObject.size` para determinar si un objeto compartido ya existe, como se muestra en el siguiente ejemplo:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

El código anterior utiliza el parámetro `size` para determinar si ya existe la instancia de objeto compartido con el nombre especificado. Si se prueba el siguiente código, se observará que, cada vez que se ejecuta, se vuelve a crear el objeto compartido. Para guardar un objeto compartido en el disco duro del usuario, debe llamarse de forma explícita al método `SharedObject.flush()`, como se muestra en el siguiente ejemplo:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

Cuando se utiliza el método `flush()` para escribir objetos compartidos en el disco duro de un usuario, es necesario comprobar si el usuario ha desactivado de forma explícita el almacenamiento local a través del Administrador de configuración de Flash Player (www.macromedia.com/support/documentation/es/flashplayer/help/settings_manager07.html), tal y como se muestra en el siguiente ejemplo:

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

Para recuperar valores de un objeto compartido, se debe especificar el nombre de la propiedad en la propiedad `data` del objeto compartido. Por ejemplo, si se ejecuta el siguiente código, Flash Player mostrará los minutos transcurridos desde que se creó la instancia de SharedObject:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
var ageMS:Number = new Date().time - so.data.now;
trace("SharedObject was created " + Number(ageMS / 1000 / 60).toPrecision(2) + " minutes ago");
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

La primera vez que se ejecuta el código anterior, se creará una instancia de SharedObject denominada `test`, que tendrá un tamaño inicial de 0 bytes. Dado que el tamaño inicial es 0 bytes, la sentencia `if` da como resultado `true` y se añade una nueva propiedad denominada `now` al objeto compartido local. La antigüedad del objeto compartido se calcula restando el valor de la propiedad `now` de la hora actual. Cada vez que vuelve a ejecutarse el código anterior, el tamaño del objeto compartido debe ser mayor que 0 y el código averigua cuántos minutos han transcurrido desde que se creó el objeto compartido.

Visualización del contenido de un objeto compartido

Los valores se almacenan en los objetos compartidos dentro de la propiedad `data`. Puede repetir indefinidamente todos los valores de una instancia de objeto compartido utilizando un bucle `for..in`, tal y como se muestra en el siguiente ejemplo:

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
    trace(i + ":\t" + so.data[i]);
}
```

Creación de un objeto SharedObject seguro

Cuando se crea una instancia de SharedObject local o remota mediante `getLocal()` o `getRemote()`, hay un parámetro opcional denominado `secure` que determina si el acceso a este objeto compartido queda restringido a los archivos SWF enviados a través de una conexión HTTPS. Si el valor de este parámetro es `true` y el archivo SWF se envía a través de HTTPS, Flash Player crea un nuevo objeto compartido seguro u obtiene una referencia a un objeto compartido seguro existente. Sólo pueden leer o escribir en este objeto compartido seguro los archivos SWF enviados a través de HTTPS que llamen a `SharedObject.getLocal()` con el parámetro secure establecido en `true`. Si el valor de este parámetro es `false` y el archivo SWF se envía a través de HTTPS, Flash Player crea un nuevo objeto compartido u obtiene una referencia a un objeto compartido existente.

Los archivos SWF enviados mediante conexiones no HTTPS pueden leer o escribir en este objeto compartido. Si un archivo SWF se ha enviado a través de una conexión no HTTPS y se intenta establecer este parámetro en `true`, no se podrá crear un nuevo objeto compartido (o acceder a un objeto compartido seguro creado anteriormente), se emitirá un error y el objeto compartido se establecerá en `null`. Si se intenta ejecutar el siguiente fragmento de código desde una conexión no HTTPS, el método `SharedObject.getLocal()` emitirá un error:

```
try
{
    var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
    trace("Unable to create SharedObject.");
}
```

Independientemente del valor de este parámetro, los objetos compartidos creados aumentan el espacio en disco total permitido en el dominio.

Trabajo con archivos de datos

Un objeto FileReference representa un archivo de datos en un equipo cliente o servidor. Los métodos de la clase FileReference permiten que la aplicación cargue y guarde archivos localmente, y que pueda transferir datos de archivo en servidores remotos.

La clase FileReference proporciona dos enfoques distintos para cargar, transferir y guardar archivos de datos. Desde su introducción, la clase FileReference incluye el método `browse()` para que el usuario pueda seleccionar un archivo, el método `upload()` para transferir los datos del archivo a un servidor remoto y el método `download()` para recuperar los datos desde el servidor y guardarlos en un archivo local. Desde Flash Player 10 y Adobe AIR 1.5, la clase FileReference tiene métodos `load()` y `save()` que permiten acceder a los archivos locales directamente y guardarlos. El uso de dichos métodos es similar a los métodos del mismo nombre incluidos en las clases URLLoader y Loader. En esta sección se tratan ambos usos de la clase FileReference.

Nota: *el motor de ejecución de AIR proporciona clases adicionales (en el paquete flash.filesystem) para trabajar con archivos y el sistema de archivos locales. Las clases de flash.filesystem ofrecen más funciones que la clase FileReference, pero sólo se admiten en AIR, no en Flash Player.*

Clase FileReference

Cada objeto FileReference hace referencia a un único archivo de datos del equipo. Las propiedades de la clase FileReference contienen la siguiente información referente al archivo: tamaño, tipo, nombre, extensión, creador, fecha de creación y fecha de modificación.

Nota: *la propiedad* `creator` *sólo se utiliza en Mac OS. En todas las demás plataformas se devuelve* `null`*.*

Nota: *la propiedad* `extension` *sólo se admite en el motor de ejecución de AIR.*

Se puede crear una instancia de la clase FileReference de una de estas dos formas:

- Con el operador `new`, como se puede ver en el siguiente código:

```
import flash.net.FileReference;
var fileRef:FileReference = new FileReference();
```

- Llame al método `FileReferenceList.browse()`, que abre un cuadro de diálogo y solicita al usuario que seleccione uno o varios archivos para cargar. Seguidamente, crea un conjunto de objetos FileReference si el usuario selecciona correctamente uno o varios archivos.

Una vez creado el objeto FileReference, puede hacer lo siguiente:

- Llamar al método `FileReference.browse()`, que abre un cuadro de diálogo y solicita al usuario que seleccione un solo archivo del sistema de archivos locales. Esto suele realizarse antes de volver a llamar al método `FileReference.upload()` para cargar el archivo en un servidor remoto o antes de llamar al método `FileReference.load()` para abrir un archivo local.
- Llamada al método `FileReference.download()`. Esto abre un cuadro de diálogo donde el usuario puede seleccionar una ubicación para guardar el nuevo archivo. Posteriormente, descarga datos del servidor y los guarda en el nuevo archivo.
- Llamada al método `FileReference.load()`. Este método comienza a cargar datos desde un archivo seleccionado previamente mediante el método `browse()`. No es posible llamar al método `load()` hasta que no finaliza la operación de `browse()` (el usuario selecciona un archivo).
- Llamada al método `FileReference.save()`. Este método abre un cuadro de diálogo y pide al usuario que seleccione una ubicación de archivo única en el sistema de archivos local. Seguidamente, guarda los datos en la ubicación especificada.

Nota: *sólo se puede realizar una acción* `browse()`*,* `download()` *o* `save()` *a la vez, ya que no se pueden abrir varios cuadros de diálogo de forma simultánea.*

Las propiedades del objeto FileReference, como `name`, `size` o `modificationDate`) no se rellenan hasta que se produce una de las situaciones siguientes:

- Se ha llamado al método `FileReference.browse()` o `FileReferenceList.browse()` y el usuario ha seleccionado un archivo desde el cuadro de diálogo.
- Se ha llamado al método `FileReference.download()` y el usuario ha especificado una nueva ubicación del archivo desde el cuadro de diálogo.

Nota: *cuando se realiza una descarga, sólo la propiedad* `FileReference.name` *se llena antes de finalizar la descarga. Una vez finalizada la descarga, todas las propiedades están disponibles.*

Durante la llamada a los métodos `FileReference.browse()`, `FileReferenceList.browse()`, `FileReference.download()`, `FileReference.load()` o `FileReference.save()`, la mayoría de reproductores continúan ejecutando el archivo SWF, incluidas la distribución de eventos y la ejecución del código.

Para las operaciones de carga y descarga, un archivo SWF sólo puede acceder a archivos de su propio dominio, incluidos los dominios especificados en un archivo de política. Es preciso colocar un archivo de política en el servidor que contiene el archivo si dicho servidor no se encuentra en el mismo dominio que el archivo SWF que inicia la carga o la descarga.

Carga de datos desde archivos

El método `FileReference.load()` permite cargar datos en la memoria local desde un archivo local. El código debe llamar primero al método `FileReference.browse()` para que el usuario pueda seleccionar un archivo para cargarlo.

El método `FileReference.load()` devuelve un valor inmediatamente después de la llamada, pero los datos que se cargan no están disponibles hasta un tiempo después. El objeto FileReference distribuye eventos para invocar métodos de detectores en cada paso del proceso de carga.

El objeto FileReference distribuye los siguientes eventos durante el proceso de carga.

- Evento `open` (`Event.OPEN`): se distribuye cuando se inicia la operación de carga.
- Evento `progress` (`ProgressEvent.PROGRESS`): se distribuye periódicamente a medida que se leen bytes de datos del archivo.
- Evento `complete` (`Event.COMPLETE`): se distribuye cuando la operación de carga finaliza correctamente.
- Evento `ioError` (`IOErrorEvent.IO_ERROR`): se distribuye si falla el proceso de carga debido a un error de entrada/salida durante la apertura o la lectura de los datos en el archivo.

Cuando el objeto FileReference distribuye el evento complete, es posible acceder a los datos cargados como un elemento ByteArray en la propiedad `data` del objeto FileReference.

El siguiente ejemplo muestra cómo solicitar al usuario que seleccione un archivo y cargue en la memoria los datos de dicho archivo:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample1 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample1()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
```

```
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            trace(fileRef.data);
        }

        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

El código de ejemplo primero crea el objeto FileReference llamado `fileRef` y, a continuación, llama a su método `browse()`. Esto abre un cuadro de diálogo que solicita al usuario que seleccione un archivo. Cuando se selecciona un archivo, se invoca el método `onFileSelected()`. Este método añade detectores para los eventos `progress` y `complete` y, seguidamente, llama al método `load()` del objeto FileReference. Los otros métodos de controladores del ejemplo simplemente producen mensajes que informan sobre el progreso de la operación de carga. Cuando finaliza la carga, la aplicación muestra el contenido del archivo cargado mediante el método `trace()`.

Guardar datos en archivos locales

El método `FileReference.save()` permite guardar datos en un archivo local. Comienza abriendo un cuadro de diálogo para que el usuario pueda introducir un nuevo nombre de archivo y una ubicación en la que guardarlo. Una vez seleccionado el nombre de archivo y la ubicación, los datos se escriben en el nuevo archivo. Una vez guardado correctamente el archivo, las propiedades del objeto FileReference se llenan con las propiedades del archivo local.

Nota: *el código sólo debe llamar al método `FileReference.save()` como respuesta a un evento de usuario (por ejemplo, un clic del ratón o la pulsación de una tecla). En caso contrario, se emite un error.*

El método `FileReference.save()` devuelve un valor inmediatamente después de recibir la llamada. Seguidamente, el objeto FileReference distribuye eventos para invocar métodos de detectores en cada paso del proceso grabación del archivo.

El objeto FileReference distribuye los siguientes eventos durante el proceso de grabación del archivo:

- Evento `select` (`Event.SELECT`): se distribuye cuando el usuario especifica la ubicación y el nombre de archivo del nuevo archivo que se va a guardar.
- Evento `cancel` (`Event.CANCEL`): se distribuye cuando el usuario hace clic en el botón Cancelar del cuadro de diálogo.
- Evento `open` (`Event.OPEN`): se distribuye cuando se inicia la operación de grabación.
- Evento `progress` (`ProgressEvent.PROGRESS`): se distribuye periódicamente a medida que los bytes de datos se guardan en el archivo.
- Evento `complete` (`Event.COMPLETE`): se distribuye cuando la operación de grabación finaliza correctamente.
- Evento `ioError` (`IOErrorEvent.IO_ERROR`): se distribuye si el proceso de grabación falla debido a un error de entrada/salida al intentar guardar los datos en el archivo.

El tipo de objeto transferido en el parámetro `data` del método `FileReference.save()` determina el modo en el que se escriben los datos en el archivo:

- Si es un valor String, se guarda como un archivo de texto con codificación UTF-8.
- Si es un objeto XML, se escribe en un archivo en formato XML (conservando todo el formato).
- Si es un objeto Array, su contenido se escribe directamente en el archivo sin conversión alguna.
- Si es cualquier otro tipo de objeto, el método `FileReference.save()` llama al método `toString()` del objeto y guarda el valor String resultante en un archivo de texto UTF-8. Si no se puede llamar al método `toString()` del objeto, se emite un error.

Si el valor del parámetro `data` es `null`, se emite un error.

El siguiente código amplía el ejemplo anterior para el método `FileReference.load()`. Una vez leídos los datos del archivo, este ejemplo solicita al usuario que especifique un nombre de archivo y, a continuación, guarda los datos en un nuevo archivo:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample2 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample2()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
```

```
public function onFileSelected(evt:Event):void
{
    fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
    fileRef.addEventListener(Event.COMPLETE, onComplete);
    fileRef.load();
}

public function onProgress(evt:ProgressEvent):void
{
    trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
}
public function onCancel(evt:Event):void
{
    trace("The browse request was canceled by the user.");
}
public function onComplete(evt:Event):void
{
    trace("File was successfully loaded.");
    fileRef.removeEventListener(Event.SELECT, onFileSelected);
    fileRef.removeEventListener(ProgressEvent.PROGRESS, onProgress);
    fileRef.removeEventListener(Event.COMPLETE, onComplete);
    fileRef.removeEventListener(Event.CANCEL, onCancel);
    saveFile();
}
public function saveFile():void
{
    fileRef.addEventListener(Event.SELECT, onSaveFileSelected);
    fileRef.save(fileRef.data,"NewFileName.txt");
}

public function onSaveFileSelected(evt:Event):void
{
    fileRef.addEventListener(ProgressEvent.PROGRESS, onSaveProgress);
    fileRef.addEventListener(Event.COMPLETE, onSaveComplete);
    fileRef.addEventListener(Event.CANCEL, onSaveCancel);
}

public function onSaveProgress(evt:ProgressEvent):void
{
    trace("Saved " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
}

public function onSaveComplete(evt:Event):void
{
    trace("File saved.");
    fileRef.removeEventListener(Event.SELECT, onSaveFileSelected);
```

```
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.removeEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.removeEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveCancel(evt:Event):void
        {
            trace("The save request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

Cuando todos los datos se han cargado desde el archivo, se invoca el método `onComplete()`. El método `onComplete()` elimina los detectores de los eventos de carga y seguidamente llama al método `saveFile()`. El método `saveFile()` llama al método `FileReference.save()`. Éste abre un nuevo cuadro de diálogo para que el usuario pueda introducir un nuevo nombre de archivo y ubicación para guardarlo. Los métodos de detectores de eventos restantes realizan el seguimiento del progreso de grabación del archivo hasta que finaliza.

Cargar archivos en un servidor

Para cargar archivos en un servidor, hay que llamar primero al método `browse()` para permitir que un usuario seleccione uno o varios archivos. A continuación, cuando se llama al método `FileReference.upload()`, el archivo seleccionado se transfiere al servidor. Si el usuario ha seleccionado varios archivos con el método `FileReferenceList.browse()`, Flash Player crea un conjunto de archivos seleccionados denominado `FileReferenceList.fileList`. A continuación, se puede utilizar el método `FileReference.upload()` para cargar cada archivo de forma individual.

Nota: *la utilización del método `FileReference.browse()` permite cargar únicamente archivos individuales. Para permitir que un usuario cargue varios archivos, es necesario utilizar el método `FileReferenceList.browse()`.*

De forma predeterminada, el cuadro de diálogo del selector de archivos del sistema permite a los usuarios seleccionar cualquier tipo de archivo del equipo local, aunque los desarrolladores pueden especificar uno o varios filtros personalizados de tipo de archivo. Para ello, deben utilizar la clase FileFilter y pasar un conjunto de instancias de filtro de archivos al método `browse()`:

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg;
*.jpeg; *.gif; *.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

Cuando el usuario ha seleccionado los archivos y ha hecho clic en el botón Abrir en el selector de archivos del sistema, se distribuye el evento `Event.SELECT`. Si se ha utilizado el método `FileReference.browse()` para seleccionar un archivo para cargarlo, es necesario utilizar el siguiente código para enviar el archivo a un servidor Web:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
    var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm")
    try
    {
        fileRef.upload(request);
    }
    catch (error:Error)
    {
        trace("Unable to upload file.");
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

Para enviar datos al servidor con el método `FileReference.upload()`, se pueden utilizar las propiedades `URLRequest.method` y `URLRequest.data` para enviar variables con los métodos `POST` o `GET`.

Cuando se intenta cargar un archivo con el método `FileReference.upload()`, se puede distribuir los siguientes eventos:

- Evento `open` (`Event.OPEN`): se distribuye cuando se inicia la operación de carga.
- Evento `progress` (`ProgressEvent.PROGRESS`): se distribuye periódicamente a medida que se cargan los bytes de datos del archivo.
- Evento `complete` (`Event.COMPLETE`): se distribuye cuando la operación de carga finaliza correctamente.
- Evento `httpStatus` (`HTTPStatusEvent.HTTP_STATUS`): se distribuye cuando el proceso de carga falla debido a un error HTTP.
- Evento `httpResponseStatus` (`HTTPStatusEvent.HTTP_RESPONSE_STATUS`): se distribuye si una llamada a los métodos `upload()` o `uploadUnencoded()` intenta acceder a los datos mediante HTTP y Adobe AIR puede detectar y devolver el código de estado de la petición.
- Evento `securityError` (`SecurityErrorEvent.SECURITY_ERROR`): se distribuye cuando se produce un error en la carga debido a una infracción de la seguridad.
- Evento `uploadCompleteData` (`DataEvent.UPLOAD_COMPLETE_DATA`): se distribuye cuando se han recibido datos del servidor tras una carga correcta.
- Evento `ioError` (`IOErrorEvent.IO_ERROR`): se distribuye si falla el proceso de carga por cualquiera de los siguientes motivos:
  - Se produce un error de entrada/salida mientras Flash Player lee, escribe o transmite el archivo.

- El archivo SWF intenta cargar un archivo en un servidor que requiere autenticación, por ejemplo, un nombre de usuario y una contraseña. Durante la carga, Flash Player no proporciona un medio para que los usuarios introduzcan contraseñas.
- El parámetro `url` contiene un protocolo no válido. El método `FileReference.upload()` debe utilizar HTTP o HTTPS.

Flash Player no ofrece compatibilidad total con los servidores que requieren autenticación. Únicamente los archivos SWF que se ejecutan en un navegador y emplean el plugin de navegador o un control Microsoft ActiveX®, pueden mostrar un cuadro de diálogo que indique al usuario que introduzca un nombre de usuario y una contraseña para la autenticación y, además, sólo para descargas. La transferencia de archivos no se producirá correctamente en las cargas que utilicen el plug-in o el control ActiveX, o en las cargas y descargas que utilicen el reproductor autónomo o externo.

Si se crea un script de servidor en ColdFusion para aceptar una carga de archivos desde Flash Player, se puede utilizar un código similar al siguiente:

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#"
nameconflict="OVERWRITE" />
```

Este código de ColdFusion carga el archivo enviado por Flash Player y lo guarda en el mismo directorio que la plantilla de ColdFusion, sobrescribiendo cualquier archivo con el mismo nombre. El código anterior muestra la cantidad mínima de código necesario para aceptar una carga de archivos; este script no debería utilizarse en un entorno de producción. Lo ideal sería añadir validación de datos para garantizar que los usuarios sólo cargan tipos de archivos aceptados como, por ejemplo, una imagen, en lugar de un script de servidor potencialmente peligroso.

El siguiente código muestra cargas de archivos mediante PHP e incluye validación de datos. El script limita en 10 el número de archivos cargados en el directorio de carga, garantiza que el tamaño de archivo es inferior a 200 KB, y sólo permite cargar y guardar archivos JPEG, GIF o PNG en el sistema de archivos.

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
    move_uploaded_file($_FILES['Filedata']['tmp_name'],
"./temporary/".$_FILES['Filedata']['name']);
    $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
    if ($type == 1 || $type == 2 || $type == 3)
    {
        rename("./temporary/".$_FILES['Filedata']['name'],
"./images/".$_FILES['Filedata']['name']);
    }
    else
    {
        unlink("./temporary/".$_FILES['Filedata']['name']);
    }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory))
{
    array_push($files, array('./images/'.$file, filectime('./images/'.$file)));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
```

```
    $files_to_delete = array_splice($files, 0, count($files) - $MAXIMUM_FILE_COUNT);
    for ($i = 0; $i < count($files_to_delete); $i++)
    {
        unlink($files_to_delete[$i][0]);
    }
}
print_r($files);
closedir($directory);

function sorter($a, $b)
{
    if ($a[1] == $b[1])
    {
        return 0;
    }
    else
    {
        return ($a[1] < $b[1]) ? -1 : 1;
    }
}
?>
```

Se pueden pasar variables adicionales al script de carga mediante el método de petición `POST` o `GET`. Para enviar variables adicionales de `POST` al script de carga, se puede utilizar el siguiente código:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
    var params:URLVariables = new URLVariables();
    params.date = new Date();
    params.ssid = "94103-1394-2345";
    var request:URLRequest = new
URLRequest("http://www.yourdomain.com/FileReferenceUpload/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    request.data = params;
    fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

El ejemplo anterior crea un nuevo objeto URLVariables que se pasa al script del servidor remoto. En versiones anteriores de ActionScript, se podían pasar variables al script de carga del servidor pasando valores en la cadena de consulta. ActionScript 3.0 permite pasar variables al script remoto con un objeto URLRequest, que permite pasar datos mediante el método `POST` o `GET` que, a su vez, hace más sencillo pasar grandes conjuntos de datos. Para especificar si las variables se pasan con el método de petición `GET` o `POST`, se puede establecer la propiedad `URLRequest.method` en `URLRequestMethod.GET` o `URLRequestMethod.POST`, respectivamente.

ActionScript 3.0 también permite sustituir el nombre del campo de archivo de carga predeterminado, `Filedata`, proporcionando un segundo parámetro al método `upload()`, como se muestra en el ejemplo anterior (que sustituía el valor predeterminado `Filedata` por `Custom1`).

De forma predeterminada, Flash Player no intentará enviar una carga de prueba. Para sustituir este comportamiento, se puede pasar el valor `true` como tercer parámetro al método `upload()`. El objetivo de la carga de prueba es comprobar si la carga de archivos real se realizará correctamente, así como la autenticación del servidor, en caso de ser necesaria.

Nota: *actualmente, la carga de prueba sólo puede realizarse en Flash Player basado en Windows.*

El script de servidor que gestiona la carga de archivos espera una petición HTTP `POST` con los siguientes elementos:

- `Content-Type` con un valor `multipart/form-data`.
- `Content-Disposition` con un atributo `name` establecido en "`Filedata`" y un atributo `filename` establecido en el nombre del archivo original. Se puede especificar un atributo `name` personalizado pasando un valor para el parámetro `uploadDataFieldName` en el método `FileReference.upload()`.
- El contenido binario del archivo.

A continuación se muestra un ejemplo de petición HTTP `POST`:

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

Test File
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

La siguiente petición de ejemplo HTTP `POST` envía tres variables `POST`: `api_sig`, `api_key` y `auth_token` y utiliza un valor de nombre de campo de datos de carga personalizado de "`photo`":

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

Descarga de archivos de un servidor

Para permitir que los usuarios descarguen archivos de un servidor, se utiliza el método `FileReference.download()`, que utiliza dos parámetros `request` y `defaultFileName`. El primer parámetro es el objeto URLRequest que contiene el URL del archivo para descargar. El segundo parámetro es opcional y permite especificar un nombre de archivo predeterminado que aparece en el cuadro de diálogo del archivo descargado. Si se omite el segundo parámetro, `defaultFileName`, se utiliza el nombre de archivo del URL especificado.

El siguiente código descarga un archivo denominado index.xml desde el mismo directorio que el documento SWF:

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

Para establecer currentnews.xml como nombre predeterminado, en lugar de index.xml, es necesario especificar el parámetro `defaultFileName`, como se muestra en el siguiente fragmento de código:

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

Cambiar el nombre de un archivo puede ser muy útil si el nombre de archivo del servidor no es intuitivo o ha sido generado por el servidor. También se recomienda especificar de forma explícita el parámetro `defaultFileName` cuando se descarga un archivo mediante un script de servidor, en lugar de descargarlo directamente. Por ejemplo, es necesario especificar el parámetro `defaultFileName` si se dispone de un script de servidor que descarga archivos específicos en función de las variables de URL que se le pasan. De lo contrario, el nombre predeterminado del archivo descargado es el nombre del script de servidor.

Es posible enviar datos al servidor mediante el método `download()` añadiendo parámetros al URL, para que los analice el script de servidor. El siguiente fragmento de código ActionScript 3.0 descarga un documento en función de los parámetros pasados a un script de ColdFusion:

```
package
{
    import flash.display.Sprite;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.net.URLRequestMethod;
    import flash.net.URLVariables;

    public class DownloadFileExample extends Sprite
    {
        private var fileToDownload:FileReference;
        public function DownloadFileExample()
        {
            var request:URLRequest = new URLRequest();
            request.url = "http://www.[yourdomain].com/downloadfile.cfm";
            request.method = URLRequestMethod.GET;
            request.data = new URLVariables("id=2");
            fileToDownload = new FileReference();
            try
            {
                fileToDownload.download(request, "file2.txt");
            }
            catch (error:Error)
            {
                trace("Unable to download file.");
            }
        }
    }
}
```

El siguiente código muestra el script ColdFusion, download.cfm, que descarga uno de los dos archivos del servidor, dependiendo del valor de una variable de URL:

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
    <cfcase value="2">
        <cfcontent type="text/plain" file="#ExpandPath('two.txt')#" deletefile="No" />
    </cfcase>
    <cfdefaultcase>
        <cfcontent type="text/plain" file="#ExpandPath('one.txt')#" deletefile="No" />
    </cfdefaultcase>
</cfswitch>
```

Clase FileReferenceList

La clase FileReferenceList permite al usuario seleccionar uno o varios archivos para cargarlos en un script de servidor. La carga de archivos se controla mediante el método `FileReference.upload()`, al que es necesario llamar en cada archivo que selecciona el usuario.

El siguiente código crea dos objetos FileFilter (`imageFilter` y `textFilter`) y los pasa en un conjunto al método `FileReferenceList.browse()`. Como consecuencia, el cuadro de diálogo del archivo del sistema operativo muestra dos filtros de tipos de archivo posibles.

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg, *.gif, *.png)",
"*.jpg; *.jpeg; *.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
    var success:Boolean = fileRefList.browse(new Array(imageFilter, textFilter));
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
```

Permitir al usuario seleccionar y cargar uno o varios archivos mediante la clase FileReferenceList equivale a utilizar `FileReference.browse()` para seleccionar archivos, aunque FileReferenceList permite seleccionar más de un archivo. Para cargar varios archivos es necesario actualizar cada uno de los archivos seleccionados mediante `FileReference.upload()`, como se muestra en el siguiente código:

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm");
    var file:FileReference;
    var files:FileReferenceList = FileReferenceList(event.target);
    var selectedFileArray:Array = files.fileList;
    for (var i:uint = 0; i < selectedFileArray.length; i++)
    {
        file = FileReference(selectedFileArray[i]);
        file.addEventListener(Event.COMPLETE, completeHandler);
        try
        {
            file.upload(request);
        }
        catch (error:Error)
        {
            trace("Unable to upload files.");
        }
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

Dado que el evento `Event.COMPLETE` se añade a cada objeto FileReference individual del conjunto, Flash Player llama al método `completeHandler()` cuando finaliza la carga de cada uno de los archivos.

Ejemplo: Generación de un cliente Telnet

En el ejemplo de Telnet se muestran las técnicas para conectar con un servidor remoto y transmitir datos con la clase Socket. El ejemplo ilustra las técnicas siguientes:

- Creación de un cliente Telnet personalizado mediante la clase Socket
- Envío de texto al servidor remoto mediante un objeto ByteArray
- Gestión de datos recibidos de un servidor remoto

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Telnet se encuentran en la carpeta Samples/Telnet. La aplicación consta de los siguientes archivos:

Archivo	Descripción
TelnetSocket.fla o TelnetSocket.mxml	El archivo principal de la aplicación formado por la interfaz de usuario de Flex (MXML) o de Flash (FLA).
TelnetSocket.as	Clase de documento que proporciona la lógica de la interfaz de usuario (sólo Flash).
com/example/programmingas3/Telnet/Telnet.as	Proporciona la funcionalidad del cliente Telnet para la aplicación como, por ejemplo, la conexión a un servidor remoto y el envío, recepción y visualización de datos.

Información general de la aplicación de socket Telnet

El archivo principal TelnetSocket.mxml es responsable de crear la interfaz de usuario de toda la aplicación.

Además de la interfaz de usuario, este archivo también define dos métodos, `login()` y `sendCommand()`, para conectar el usuario al servidor especificado.

A continuación se muestra el código ActionScript del archivo de la aplicación principal:

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
    telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
    console.title = "Connecting to " + serverName.text + ":" + portNumber.text;
    console.enabled = true;
}
private function sendCommand():void
{
    var ba:ByteArray = new ByteArray();
    ba.writeMultiByte(command.text + "\n", "UTF-8");
    telnetClient.writeBytesToSocket(ba);
    command.text = "";
}
```

La primera línea de código importa la clase Telnet del paquete com.example.programmingas.socket personalizado. La segunda línea de código declara una instancia de la clase Telnet, `telnetClient`, que se inicializará posteriormente mediante el método `connect()`. A continuación, se declara el método `connect()` e inicializa la variable `telnetClient` declarada previamente. Este método pasa el nombre del servidor Telnet especificado por el usuario, el puerto del servidor Telnet y una referencia a un componente TextArea de la lista de visualización, que se utiliza para mostrar las respuestas de texto del servidor de socket. Las dos últimas líneas del método `connect()` establecen la propiedad `title` de Panel y activan el componente Panel, que permite al usuario enviar datos al servidor remoto. El método final del archivo principal de la aplicación, `sendCommand()`, se utiliza para enviar los comandos del usuario al servidor remoto como un objeto ByteArray.

Información general de la clase Telnet

La clase Telnet es la responsable de conectar con el servidor Telnet remoto, y de enviar y recibir datos.

La clase Telnet declara las siguientes variables privadas:

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

La primera variable, `serverURL`, contiene la dirección del servidor, especificada por el usuario, con la que se va a conectar.

La segunda variable, `portNumber`, es el número de puerto en el que se está ejecutando el servidor Telnet. De forma predeterminada, el servicio Telnet se ejecuta en el puerto 23.

La tercera variable, `socket`, es una instancia de Socket que intentará conectar con el servidor definido por las variables `serverURL` y `portNumber`.

La cuarta variable, `ta`, es una referencia a una instancia del componente TextArea en el escenario. Este componente se utiliza para mostrar las respuestas del servidor Telnet remoto o cualquier posible mensaje de error.

La última variable, `state`, es un valor numérico que se utiliza para determinar las opciones que admite el cliente Telnet.

Tal y como se ha visto previamente, se llama a la función constructora de la clase Telnet a través del método `connect()` en el archivo de la aplicación principal.

El constructor Telnet adopta tres parámetros: `server`, `port` y `output`. Los parámetros `server` y `port` especifican el nombre de servidor y el número de puerto donde se ejecuta el servidor Telnet. El parámetro final, `output`, es una referencia a una instancia del componente TextArea en el escenario, donde los usuarios verán la salida del servidor.

```
public function Telnet(server:String, port:int, output:TextArea)
{
    serverURL = server;
    portNumber = port;
    ta = output;
    socket = new Socket();
    socket.addEventListener(Event.CONNECT, connectHandler);
    socket.addEventListener(Event.CLOSE, closeHandler);
    socket.addEventListener(ErrorEvent.ERROR, errorHandler);
    socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
    socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
    Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
    try
    {
        msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
        socket.connect(serverURL, portNumber);
    }
    catch (error:Error)
    {
        msg(error.message + "\n");
        socket.close();
    }
}
```

Escritura de datos en un socket

Para escribir datos en una conexión de socket, hay que llamar a cualquiera de los métodos de la clase Socket (como `writeBoolean()`, `writeByte()`, `writeBytes()` o `writeDouble()`) y luego vaciar los datos en el búfer de salida con el método `flush()`. En el servidor Telnet, los datos se escriben en la conexión de socket mediante el método `writeBytes()`, que utiliza el conjunto de bytes como parámetro y lo envía al búfer de salida. La sintaxis del método `writeBytesToSocket()` es la siguiente:

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

Este método se llama mediante el método `sendCommand()` del archivo de la aplicación principal.

Visualización de mensajes del servidor de socket

Cuando se recibe un mensaje del servidor de socket o se produce un evento, se llama al método personalizado `msg()`. Este método añade una cadena a TextArea en el escenario y llama a un método personalizado `setScroll()`, que desplaza el componente TextArea hasta la parte más baja. La sintaxis del método `msg()` es la siguiente:

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

Si el contenido del componente TextArea no se desplaza automáticamente, el usuario deberá arrastrar manualmente las barras de desplazamiento en el área de texto para ver la última respuesta del servidor.

Desplazamiento de un componente TextArea

El método `setScroll()` contiene una sola línea de código ActionScript que desplaza verticalmente el contenido del componente TextArea para que el usuario pueda ver la última línea del texto devuelto. El siguiente fragmento de código muestra el método `setScroll()`:

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

Este método establece la propiedad `verticalScrollPosition`, que es el número de línea de la fila superior de caracteres que se visualiza, con el valor de la propiedad `maxVerticalScrollPosition`.

Ejemplo: Carga y descarga de archivos

El ejemplo FileIO muestra las técnicas para cargar y descargar archivos en Flash Player. Estas técnicas son:

- Descarga de archivos en el equipo de un usuario
- Carga de archivos del equipo de un usuario en un servidor
- Cancelación de una descarga en curso
- Cancelación de una carga en curso

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación FileIO se encuentran en la carpeta Samples/FileIO. La aplicación consta de los siguientes archivos:

Archivo	Descripción
FileIO.fla o FileIO.mxml	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/fileio/FileDownload.as	Una clase que incluye métodos para descargar archivos de un servidor.
com/example/programmingas3/fileio/FileUpload.as	Una clase que incluye métodos para cargar archivos en un servidor.

Información general de la aplicación FileIO

La aplicación FileIO contiene la interfaz de usuario que permite a un usuario cargar o descargar archivos con Flash Player. En primer lugar, la aplicación define un par de componentes personalizados, FileUpload y FileDownload, que se encuentran en el paquete com.example.programmingas3.fileio. Cuando cada componente personalizado distribuye su evento `contentComplete`, se llama al método `init()` del componente y pasa referencias a una instancia de los componentes ProgressBar y Button, que permite a los usuarios ver el progreso de la carga o descarga del archivo, o cancelar la transferencia de archivos en curso.

A continuación se muestra el código relevante del archivo FileIO.mxml (hay que tener en cuenta que en la versión de Flash, el archivo FLA contiene componentes colocados en el escenario cuyos nombres coinciden con los de los componentes Flex descritos en este paso):

```
<example:FileUpload id="fileUpload" creationComplete="fileUpload.init(uploadProgress,
cancelUpload);" />
<example:FileDownload id="fileDownload"
creationComplete="fileDownload.init(downloadProgress, cancelDownload);" />
```

El siguiente código muestra el panel de carga de archivos (Upload File), que contiene una barra de progreso y dos botones. El primero botón, `startUpload`, llama al método `FileUpload.startUpload()`, que llama al método `FileReference.browse()`. El siguiente fragmento de código corresponde al panel Upload File:

```
<mx:Panel title="Upload File" paddingTop="10" paddingBottom="10" paddingLeft="10"
paddingRight="10">
    <mx:ProgressBar id="uploadProgress" label="" mode="manual" />
    <mx:ControlBar horizontalAlign="right">
        <mx:Button id="startUpload" label="Upload..." click="fileUpload.startUpload();" />
        <mx:Button id="cancelUpload" label="Cancel" click="fileUpload.cancelUpload();"
enabled="false" />
    </mx:ControlBar>
</mx:Panel>
```

Este código coloca una instancia del componente ProgressBar y dos instancias del componente Button en el escenario. Cuando el usuario hace clic en el botón de carga (`startUpload`), se abre un cuadro de diálogo del sistema operativo, que permite al usuario seleccionar un archivo para cargarlo en un servidor remoto. El otro botón, `cancelUpload`, está desactivado de forma predeterminada; cuando un usuario inicia la carga de un archivo, el botón se activa y permite al usuario cancelar la transferencia del archivo en cualquier momento.

El código correspondiente al panel de descarga de archivos (Download File) es el siguiente:

```
<mx:Panel title="Download File" paddingTop="10" paddingBottom="10" paddingLeft="10"
paddingRight="10">
    <mx:ProgressBar id="downloadProgress" label="" mode="manual" />
    <mx:ControlBar horizontalAlign="right">
        <mx:Button id="startDownload" label="Download..."
click="fileDownload.startDownload();" />
        <mx:Button id="cancelDownload" label="Cancel" click="fileDownload.cancelDownload();"
enabled="false" />
    </mx:ControlBar>
</mx:Panel>
```

Este código es muy similar al código para cargar archivos. Cuando el usuario hace clic en el botón de descarga, (`startDownload`), se llama al método `FileDownload.startDownload()`, que inicia la descarga del archivo especificado en la variable `FileDownload.DOWNLOAD_URL`. La barra de progreso, que se actualiza a medida que avanza la descarga del archivo, muestra el porcentaje del archivo que se ha descargado. El usuario puede cancelar la descarga en cualquier momento haciendo clic en el botón `cancelDownload`, que detiene inmediatamente la descarga del archivo en curso.

Descarga de archivos de un servidor remoto

La descarga de archivos de un servidor remoto se controla mediante la clase flash.net.FileReference y la clase personalizada com.example.programmingas3.fileio.FileDownload. Cuando el usuario hace clic en el botón de descarga, Flash Player inicia la descarga del archivo especificado en la variable `DOWNLOAD_URL` de la clase FileDownload.

La clase FileDownload define cuatro variables en el paquete com.example.programmingas3.fileio, como se muestra en el siguiente código:

```
/**
 * Hard-code the URL of file to download to user's computer.
 */
private const DOWNLOAD_URL:String = "http://www.yourdomain.com/file_to_download.zip";

/**
 * Create a FileReference instance to handle the file download.
 */
private var fr:FileReference;

/**
 * Define reference to the download ProgressBar component.
 */
private var pb:ProgressBar;

/**
 * Define reference to the "Cancel" button which will immediately stop
 * the current download in progress.
 */
private var btn:Button;
```

La primera variable, `DOWNLOAD_URL`, contiene la ruta de acceso al archivo, que se descarga en el equipo del usuario cuando éste hace clic en el botón de descarga, en el archivo de la aplicación principal.

La segunda variable, `fr`, es un objeto FileReference que se inicializa en el método `FileDownload.init()` y que controla la descarga del archivo remoto en el equipo del usuario.

Las dos últimas variables, `pb` y `btn`, contienen referencias a las instancias de los componentes ProgressBar y Button en el escenario, que se inicializan con el método `FileDownload.init()`.

Inicialización del componente FileDownload

El componente FileDownload se inicializa mediante una llamada al método `init()` en la clase FileDownload. Este método utiliza dos parámetros, `pb` y `btn`, que son instancias de los componentes ProgressBar y Button, respectivamente.

El código correspondiente al método `init()` es el siguiente:

```
/**
 * Set references to the components, and add listeners for the OPEN,
 * PROGRESS, and COMPLETE events.
 */
public function init(pb:ProgressBar, btn:Button):void
{
    // Set up the references to the progress bar and cancel button,
    // which are passed from the calling script.
    this.pb = pb;
    this.btn = btn;

    fr = new FileReference();
    fr.addEventListener(Event.OPEN, openHandler);
    fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

Inicio de la descarga de archivos

Cuando el usuario hace clic en la instancia del componente Download Button en el escenario, el método `startDownload()` inicia el proceso de descarga de archivos. El siguiente fragmento de código muestra el método `startDownload()`:

```
/**
 * Begin downloading the file specified in the DOWNLOAD_URL constant.
 */
public function startDownload():void
{
    var request:URLRequest = new URLRequest();
    request.url = DOWNLOAD_URL;
    fr.download(request);
}
```

En primer lugar, el método `startDownload()` crea un nuevo objeto URLRequest y establece el URL de destino en el valor especificado por la variable `DOWNLOAD_URL`. A continuación, se llama al método `FileReference.download()` y el objeto URLRequest recién creado se pasa como un parámetro. Como consecuencia, el sistema operativo muestra un cuadro de diálogo en el equipo del usuario, donde le pide que seleccione una ubicación para guardar el documento solicitado. Cuando el usuario ha seleccionado una ubicación, se distribuye el evento `open` (`Event.OPEN`) y se llama al método `openHandler()`.

El método `openHandler()` establece el formato de texto de la propiedad `label` del componente ProgressBar y activa el botón de cancelación, que permite al usuario detener inmediatamente la descarga en curso. La sintaxis del método `openHandler()` es la siguiente:

```
/**
 * When the OPEN event has dispatched, change the progress bar's label
 * and enable the "Cancel" button, which allows the user to abort the
 * download operation.
 */
private function openHandler(event:Event):void
{
    pb.label = "DOWNLOADING %3%%";
    btn.enabled = true;
}
```

Supervisión del progreso de descarga de un archivo

Cuando un archivo se descarga desde un servidor remoto en el equipo del usuario, el evento `progress` (`ProgressEvent.PROGRESS`) se distribuye en intervalos regulares. Cuando se distribuye el evento `progress`, se llama al método `progressHandler()` y se actualiza la instancia del componente ProgressBar en el escenario. El código correspondiente al método `progressHandler()` es el siguiente:

```
/**
 * While the file is downloading, update the progress bar's status.
 */
private function progressHandler(event:ProgressEvent):void
{
    pb.setProgress(event.bytesLoaded, event.bytesTotal);
}
```

El evento progress contiene dos propiedades, `bytesLoaded` y `bytesTotal`, que se utilizan para actualizar el componente ProgressBar en el escenario. De esta forma, el usuario puede saber la cantidad de archivo que ya se ha descargado y la que queda por descargar. El usuario puede cancelar la transferencia del archivo en cualquier momento haciendo clic en el botón de cancelación situado bajo la barra de progreso.

Si el archivo se descarga correctamente, el evento `complete` (`Event.COMPLETE`) llama al método `completeHandler()`, que notifica al usuario que ha finalizado la descarga del archivo y desactiva el botón de cancelación. El código correspondiente al método `completeHandler()` es el siguiente:

```
/**
 * Once the download has completed, change the progress bar's label one
 * last time and disable the "Cancel" button since the download is
 * already completed.
 */
private function completeHandler(event:Event):void
{
    pb.label = "DOWNLOAD COMPLETE";
    btn.enabled = false;
}
```

Cancelación de una descarga de archivos

Un usuario puede cancelar una transferencia de archivos y detener la descarga de más bytes en cualquier momento haciendo clic en el botón de cancelación en el escenario. El siguiente fragmento de código muestra la cancelación de una descarga:

```
/**
 * Cancel the current file download.
 */
public function cancelDownload():void
{
    fr.cancel();
    pb.label = "DOWNLOAD CANCELLED";
    btn.enabled = false;
}
```

En primer lugar, el código detiene inmediatamente la transferencia del archivo, evitando que se descarguen más datos. A continuación, la propiedad de la etiqueta de la barra de progreso se actualiza para notificar al usuario que la descarga se ha cancelado correctamente. Finalmente, se desactiva el botón de cancelación, que evita que el usuario haga clic nuevamente en el botón hasta se intente volver a descargar el archivo.

Carga de archivos en un servidor remoto

El proceso de carga de archivos es muy similar al proceso de descarga de archivos. La clase FileUpload declara las mismas cuatro variables, como se muestra en el siguiente código:

```
private const UPLOAD_URL:String = "http://www.yourdomain.com/your_upload_script.cfm";
private var fr:FileReference;
private var pb:ProgressBar;
private var btn:Button;
```

A diferencia de la variable `FileDownload.DOWNLOAD_URL`, la variable `UPLOAD_URL` contiene el URL correspondiente al script de servidor que cargará el archivo desde el equipo del usuario. Las tres variables restantes se comportan del mismo modo que las variables correspondientes de la clase FileDownload.

Inicialización del componente FileUpload

El componente FileUpload contiene un método `init()`, al que se llama desde la aplicación principal. Este método utiliza dos parámetros, `pb` y `btn`, que son referencias a las instancias de los componentes ProgressBar y Button en el escenario. A continuación, el método `init()` inicializa el objeto FileReference definido previamente en la clase FileUpload. Finalmente, el método asigna cuatro detectores de eventos al objeto FileReference. El código correspondiente al método `init()` es el siguiente:

```
public function init(pb:ProgressBar, btn:Button):void
{
    this.pb = pb;
    this.btn = btn;

    fr = new FileReference();
    fr.addEventListener(Event.SELECT, selectHandler);
    fr.addEventListener(Event.OPEN, openHandler);
    fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

Inicio de la carga de archivos

La carga de archivos se inicia cuando el usuario hace clic en el botón de carga en el escenario, que llama al método `FileUpload.startUpload()`. Este método llama al método `browse()` de la clase FileReference, lo que provoca que se abra un cuadro de diálogo del sistema operativo, donde se pide al usuario que seleccione un archivo para cargarlo en un servidor remoto. El siguiente fragmento de código muestra el método `startUpload()`:

```
public function startUpload():void
{
    fr.browse();
}
```

Cuando el usuario selecciona un archivo para cargar, se distribuye el evento `select` (`Event.SELECT`), que provoca una llamada al método `selectHandler()`. El método `selectHandler()` crea un nuevo objeto URLRequest y establece la propiedad `URLRequest.url` en el valor de la constante `UPLOAD_URL` definida previamente en el código. Finalmente, el objeto FileReference carga el archivo seleccionado en el script de servidor especificado. El código correspondiente al método `selectHandler()` es el siguiente:

```
private function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest();
    request.url = UPLOAD_URL;
    fr.upload(request);
}
```

El código restante de la clase FileUpload es el mismo código definido en la clase FileDownload. Si un usuario desea interrumpir la descarga en cualquier momento, puede hacer clic en el botón de cancelación, que establece la etiqueta de la barra de progreso y detiene inmediatamente la transferencia del archivo. La barra de progreso se actualiza siempre que se distribuye el evento `progress` (`ProgressEvent.PROGRESS`). De forma similar, cuando ha finalizado la carga, la barra de progreso se actualiza para notificar al usuario que el archivo se ha cargado correctamente. El botón de cancelación se desactiva hasta que el usuario inicie una nueva transferencia de archivos.

Capítulo 28: Entorno del sistema del cliente

En este capítulo se explica la manera de interactuar con el sistema del usuario. Muestra la manera de determinar qué funciones se admiten y cómo crear archivos SWF multilingües mediante el editor de método de entrada (IME) instalado en el sistema del usuario (si está disponible). También muestra usos típicos de los dominios de aplicación.

Fundamentos del entorno del sistema del cliente

Introducción al entorno del sistema del cliente

Al crear aplicaciones de ActionScript más avanzadas puede que sea necesario conocer los detalles sobre los sistemas operativos de los usuarios y acceder a funciones de dichos sistemas. El entorno del sistema del cliente es una colección de clases del paquete flash.system que permiten acceder a funcionalidad de nivel de sistema como la siguiente:

- Determinar la aplicación y el dominio de seguridad en el que se está ejecutando un archivo SWF
- Determinar las características del reproductor Flash® Player del usuario de la instancia de Adobe® AIR™, como el tamaño de la pantalla (resolución) y si determinadas funcionalidades , como el audio MP3, están disponibles
- Generar sitios multilingües con el IME
- Interactuar con el contenedor de Flash Player (que puede ser una página HTML o una aplicación contenedora) o con el contenedor de AIR.
- Guardar información en el portapapeles del usuario

El paquete flash.system también incluye las clases IMEConversionMode y SecurityPanel. Estas clases contienen constantes estáticas que se utilizan con las clases IME y Security, respectivamente.

Tareas comunes en el entorno del sistema del cliente

En este capítulo se describen las siguientes tareas comunes del trabajo con el sistema del cliente mediante ActionScript:

- Determinar la cantidad de memoria que utiliza la aplicación
- Copiar texto al portapapeles del usuario
- Determinar las características del equipo del usuario, como:
  - Resolución de pantalla, color, PPP y proporción en píxeles
  - Sistema operativo
  - Compatibilidad con transmisión de flujo de audio y de vídeo, y reproducción MP3
  - Si la versión de Flash Player instalada es una versión de depuración
- Trabajo con dominios de aplicación:
  - Definir un dominio de aplicación
  - Separar el código de los archivos SWF en dominios de aplicación

- Trabajo con un IME en la aplicación:
  - Determinar si hay un IME instalado
  - Determinar y configurar el modo de conversión del IME
  - Desactivar el IME para campos de texto
  - Detectar la conversión de IME

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Sistema operativo: programa principal que se ejecuta en un equipo, con el que se ejecutan todas las demás aplicaciones, como Microsoft Windows, Mac OS X o Linux®.
- Portapapeles: contenedor del sistema operativo para texto o elementos que se copian o cortan, y desde el que se pegan elementos en aplicaciones.
- Dominio de aplicación: mecanismo para separar clases utilizadas en distintos archivos SWF de forma que, si los archivos SWF incluyen distintas clases con el mismo nombre, no se sobrescriban unas a otras.
- IME (editor de método de entrada): programa (o herramienta del sistema operativo) que se utiliza para introducir caracteres o símbolos complejos mediante un teclado estándar.
- Sistema del cliente: en términos de programación, un *cliente* es la parte de una aplicación (o una aplicación completa) que se ejecuta en un equipo individual y es utilizada por un solo usuario. El *sistema del cliente* es el sistema operativo subyacente en el equipo del usuario.

Ejecución de los ejemplos del capítulo

A medida que progresa en el estudio del capítulo, es posible que desee probar algunos de los listados de código. Todos los listados de código de este capítulo incluyen la llamada apropiada a la función `trace()` para escribir los valores que se están probando. Para probar los listados de código de este capítulo:

1. Cree un documento de Flash vacío.
2. Seleccione un fotograma clave en la línea de tiempo.
3. Abra el panel Acciones y copie el listado de código en el panel Script.
4. Ejecute el programa seleccionando Control > Probar película.

   El resultado de las funciones `trace()` del código se ve en el panel Salida.

Algunos de los últimos ejemplos de código son más complejos y se han programado como una clase. Para probar estos ejemplos:

1. Cree un documento de Flash vacío y guárdelo en el equipo.
2. Cree un nuevo archivo de ActionScript y guárdelo en el mismo directorio que el documento de Flash. El nombre del archivo debe coincidir con el nombre de la clase del listado de código. Por ejemplo, si el listado de código define una clase denominada SystemTest, use el nombre SystemTest.as para guardar el archivo de ActionScript.
3. Copie el listado de código en el archivo de ActionScript y guarde el archivo.
4. En el documento de Flash, haga clic en una parte vacía del escenario o espacio de trabajo para activar el inspector de propiedades del documento.
5. En el inspector de propiedades, en el campo Clase de documento, escriba el nombre de la clase de ActionScript que copió del texto.

6 Ejecute el programa seleccionando Control > Probar película.

Verá el resultado del ejemplo en el panel Salida.

Las técnicas para probar los listados de código de ejemplo se describen de forma detallada en "Prueba de los listados de código de ejemplo del capítulo" en la página 36.

Utilización de la clase System

La clase System contiene métodos y propiedades que permiten interactuar con el sistema operativo del usuario y obtener el consumo de memoria actual de Flash Player o AIR. Los métodos y propiedades de la clase System también permiten detectar eventos `imeComposition`, ordenar a Flash Player o AIR que cargue archivos de texto externos con la página de códigos actual del usuario o como Unicode, o establecer el contenido del portapapeles del usuario.

Obtención de datos sobre el sistema del usuario en tiempo de ejecución

Se puede comprobar el valor de la propiedad `System.totalMemory` para determinar la cantidad de memoria (en bytes) utilizada actualmente por Flash Player o AIR. Esta propiedad permite controlar el consumo de memoria y optimizar las aplicaciones a partir de los cambios de nivel de la memoria. Por ejemplo, si un efecto visual específico provoca un gran aumento de consumo de memoria, es posible que se desee modificar el efecto o eliminarlo del todo.

La propiedad `System.ime` es una referencia al editor de método de entrada (IME) instalado actualmente. Esta propiedad permite detectar eventos `imeComposition` (`flash.events.IMEEvent.IME_COMPOSITION`) mediante el método `addEventListener()`.

La tercera propiedad de la clase System es `useCodePage`. Cuando se establece `useCodePage` en `true`, Flash Player y AIR utilizan la página de códigos tradicional del sistema operativo que ejecuta el reproductor para cargar los archivos de texto externos. Si se establece esta propiedad en `false`, se indica a Flash Player o AIR que debe interpretar el archivo externo como Unicode.

Si se establece `System.useCodePage` en el valor `true`, hay que recordar que la página de códigos tradicional del sistema operativo en el que se ejecuta el reproductor debe incluir los caracteres utilizados en el archivo de texto externo para que se muestre el texto. Por ejemplo, si se carga un archivo de texto externo que contiene caracteres chinos, dichos caracteres no se visualizarán en un sistema que utilice la página de códigos de Windows en inglés, ya que dicha página de códigos no contiene caracteres chinos.

Para asegurarse de que los usuarios de todas las plataformas puedan ver los archivos de texto externos que se utilizan en sus archivos SWF, hay que codificar todos los archivos de texto externos como Unicode y establecer `System.useCodePage` en `false` de forma predeterminada. De esta forma, Flash Player y AIR interpretarán el texto como Unicode.

Copia de texto al portapapeles

La clase System incluye un método denominado `setClipboard()` que permite a Flash Player y AIR establecer el contenido del portapapeles del usuario con una cadena especificada. Por razones de seguridad, no hay un método `Security.getClipboard()`, ya que este método podría permitir a sitios malintencionados el acceso a los últimos datos copiados al portapapeles del usuario.

El código siguiente ilustra cómo se puede copiar un mensaje de error al portapapeles del usuario cuando se produce un error de seguridad. El mensaje de error puede ser útil si el usuario desea notificar un posible error con una aplicación.

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

Utilización de la clase Capabilities

La clase Capabilities permite a los desarrolladores determinar el entorno en el que se está ejecutando un archivo SWF. Se pueden utilizar diversas propiedades de la clase Capabilities para averiguar la resolución del sistema del usuario, si dicho sistema admite software de accesibilidad y el lenguaje del sistema operativo del usuario, así como la versión del reproductor Flash Player o AIR que tienen instalada en ese momento.

Se pueden comprobar los valores de las propiedades de la clase Capabilities para personalizar la aplicación de forma que funcione de forma óptima con el entorno específico del usuario. Por ejemplo, si se comprueban los valores de las propiedades `Capabilities.screenResolutionX` y `Capabilities.screenResolutionY`, se puede determinar la resolución de pantalla que utiliza el sistema del usuario y decidir qué tamaño de vídeo es el más apropiado. También se puede comprobar el valor de la propiedad `Capabilities.hasMP3` para comprobar si el sistema del usuario admite la reproducción de MP3 antes de intentar cargar un archivo MP3 externo.

El código siguiente utiliza una expresión regular para analizar la versión de Flash Player que utiliza el cliente:

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
    trace("input: " + result.input);
    trace("platform: " + result[1]);
    trace("majorVersion: " + result[2]);
    trace("minorVersion: " + result[3]);
    trace("buildNumber: " + result[4]);
    trace("internalBuildNumber: " + result[5]);
}
else
{
    trace("Unable to match RegExp.");
}
```

Si se desea enviar las características del sistema del usuario a un script de servidor de forma que se pueda almacenar la información en una base de datos, se puede utilizar el siguiente código ActionScript:

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

Utilización de la clase ApplicationDomain

El propósito de la clase ApplicationDomain es almacenar una tabla de definiciones de ActionScript 3.0. Todo el código de un archivo SWF se define como existente en un dominio de aplicación. Los dominios de aplicación se utilizan para hacer particiones de clases en el mismo dominio de seguridad. Esto permite que coexistan varias definiciones de la misma clase y que los elementos secundarios puedan reutilizar definiciones de elementos principales.

Se pueden utilizar dominios de aplicación al cargar un archivo SWF externo programado en ActionScript 3.0 mediante la API de la clase Loader. (Hay que tener en cuenta que no se pueden utilizar dominios de aplicación al cargar una imagen o un archivo SWF programado en ActionScript 1.0 o ActionScript 2.0.) Todas las definiciones de ActionScript 3.0 contenidas en la clase cargada se almacenan en el dominio de aplicación. Al cargar el archivo SWF, se puede especificar que se incluya el archivo en el mismo dominio de aplicación que el del objeto Loader, estableciendo el parámetro `applicationDomain` del objeto LoaderContext en `ApplicationDomain.currentDomain`. Al colocar el archivo SWF cargado en el mismo dominio de aplicación, se puede acceder a sus clases directamente. Esto puede resultar útil si se carga un archivo SWF que contiene medios incorporados, a los que se puede acceder a través de sus nombres de clases asociados, o si se desea acceder métodos del archivo SWF cargado, como se indica en el siguiente ejemplo:

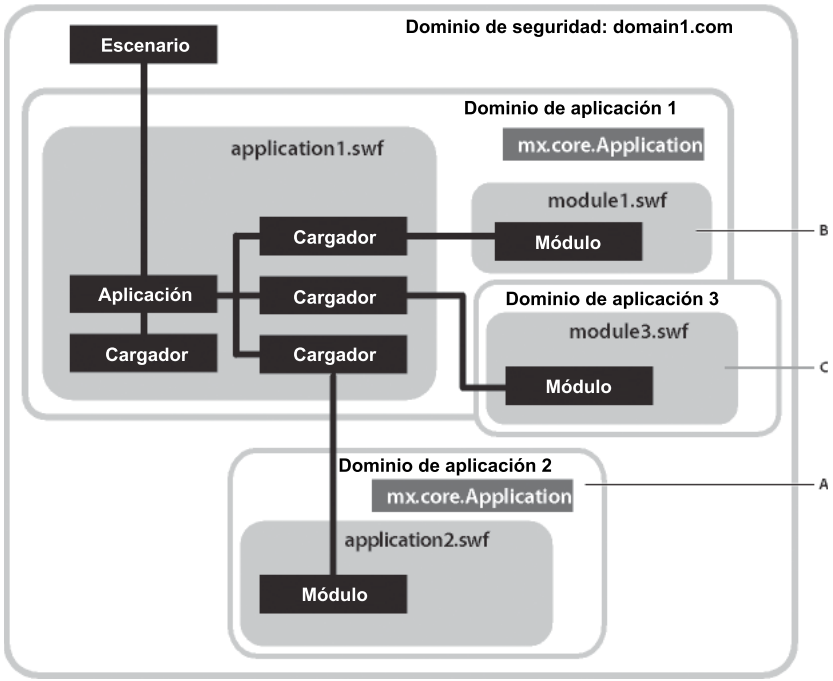
```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    public class ApplicationDomainExample extends Sprite
    {
        private var ldr:Loader;
        public function ApplicationDomainExample()
        {
            ldr = new Loader();
            var req:URLRequest = new URLRequest("Greeter.swf");
            var ldrContext:LoaderContext = new LoaderContext(false,
ApplicationDomain.currentDomain);
            ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
            ldr.load(req, ldrContext);
        }
        private function completeHandler(event:Event):void
        {
            ApplicationDomain.currentDomain.getDefinition("Greeter");
            var myGreeter:Greeter = Greeter(event.target.content);
            var message:String = myGreeter.welcome("Tommy");
            trace(message); // Hello, Tommy
        }
    }
}
```

Otras cosas que hay que tener en cuenta al trabajar con dominios de aplicación son las siguientes:

- Todo el código de un archivo SWF se define como existente en un dominio de aplicación. El *dominio* actual es el lugar en el que se ejecuta la aplicación principal. El *dominio del sistema* contiene todos los dominios de la aplicación, incluido el dominio actual, lo que significa que contiene todas las clases de Flash Player.

- Todos los dominios de aplicación tienen asociado un dominio principal, excepto el dominio del sistema. El dominio principal del dominio de aplicación principal es el dominio del sistema. Sólo es necesario definir las clases cargadas si su clase principal no las ha definido todavía. No es posible anular una definición de clase cargada con otra definición más reciente.

En el diagrama siguiente se muestra una aplicación que carga contenido de diversos archivos SWF de un solo dominio, domain1.com. En función del contenido que se cargue, se pueden utilizar distintos dominios de aplicación. El texto siguiente describe la lógica que se utiliza para establecer el dominio de aplicación apropiado para cada archivo SWF de la aplicación.



***A.** Uso A **B.** Uso B **C.** Uso C*

El archivo de aplicación principal es application1.swf. Contiene objetos Loader que cargan contenido de otros archivos SWF. En este escenario, el dominio actual es Application domain 1. Los usos A, B y C ilustran distintas técnicas para establecer el dominio de aplicación apropiado para cada archivo SWF de una aplicación.

Uso A dividir el archivo SWF secundario creando un elemento secundario del dominio del sistema. En el diagrama, el dominio de aplicación 2 se crea como un elemento secundario del dominio del sistema. El archivo application2.swf se carga en el dominio de aplicación 2 y sus definiciones de clase se dividen de las clases definidas en application1.swf.

Una aplicación de esta técnica es hacer que una aplicación antigua cargue dinámicamente una versión más reciente de la misma aplicación sin conflictos. No hay conflictos porque, aunque se usen los mismos nombres de clase, se dividen en distintos dominios de aplicación.

El código siguiente crea un dominio de aplicación que es un elemento secundario del dominio del sistema y comienza a cargar un archivo SWF que utiliza ese dominio de aplicación:

```
var appDomainA:ApplicationDomain = new ApplicationDomain();

var contextA:LoaderContext = new LoaderContext(false, appDomainA);
var loaderA:Loader = new Loader();
loaderA.load(new URLRequest("application2.swf"), contextA);
```

Uso B: añadir nuevas definiciones de clase a las definiciones de clase actuales. El dominio de aplicación de module1.swf se establece en el dominio actual (dominio de aplicación 1). Esto permite añadir nuevas definiciones de clase al conjunto actual de definiciones de clase de la aplicación. Esto se puede utilizar para una biblioteca compartida en tiempo de ejecución de la aplicación principal. El archivo SWF cargado se trata como una biblioteca remota compartida (RSL). Esta técnica se utiliza para cargar bibliotecas RSL mediante un precargador antes de que se inicie la aplicación.

El código siguiente carga un archivo SWF, estableciendo su dominio de aplicación en el dominio actual:

```
var appDomainB:ApplicationDomain = ApplicationDomain.currentDomain;

var contextB:LoaderContext = new LoaderContext(false, appDomainB);
var loaderB:Loader = new Loader();
loaderB.load(new URLRequest("module1.swf"), contextB);
```

Uso C: utilizar las definiciones de clase del elemento principal creando un nuevo dominio secundario del dominio actual. El dominio de aplicación de module3.swf es un elemento secundario del dominio actual y el elemento secundario utiliza las versiones del elemento principal de todas las clases. Una aplicación de esta técnica podría ser un módulo de una aplicación de Internet compleja de varias pantallas, cargada como un elemento secundario de la aplicación principal y que utiliza los tipos de la aplicación principal. Si hay la seguridad de que siempre se actualizarán todas las clases para ser compatibles con versiones anteriores y que la aplicación de carga es siempre más reciente que los elementos que carga, los elementos secundarios utilizarán las versiones del elemento principal. Tener un nuevo dominio de aplicación también permite descargar todas las definiciones de clase para eliminar datos innecesarios, si hay la seguridad de que no siguen existiendo referencias al archivo SWF secundario.

Esta técnica permite que los módulos cargados compartan los objetos singleton del cargador y los miembros de clase estáticos.

El código siguiente crea un nuevo dominio secundario del dominio actual y comienza a cargar un archivo SWF que utiliza ese dominio de aplicación:

```
var appDomainC:ApplicationDomain = new ApplicationDomain(ApplicationDomain.currentDomain);

var contextC:LoaderContext = new LoaderContext(false, appDomainC);
var loaderC:Loader = new Loader();
loaderC.load(new URLRequest("module3.swf"), contextC);
```

Utilización de la clase IME

La clase IME permite manipular el IME del sistema operativo en Flash Player o Adobe AIR.

También se puede utilizar código ActionScript para determinar lo siguiente:

- Si hay instalado un IME en el equipo del usuario (`Capabilities.hasIME`).
- Si el IME está activado o desactivado en el equipo del usuario (`IME.enabled`).
- El modo de conversión utilizado por el IME actual (`IME.conversionMode`).

Se puede asociar un campo de entrada de texto con un contexto de IME específico. Al cambiar de un campo de entrada a otro, también se puede cambiar el IME a Hiragana (Japonés), números de anchura completa, números de anchura media, entrada directa, etc..

Un IME permite al usuario introducir caracteres de texto en idiomas multibyte, como chino, japonés o coreano, con una codificación distinta de ASCII.

Para más información sobre los IME, consulte la documentación del sistema operativo para el que desarrolla la aplicación. Para ver recursos adicionales, consulte los siguientes sitios Web:

- http://www.msdn.microsoft.com/goglobal/
- http://developer.apple.com/documentation/
- http://www.java.sun.com/

Nota: *si un IME no está activo en el ordenador del usuario, fallarán todas las llamadas a los métodos o propiedades IME que no sean* `Capabilities.hasIME`*. Tras activar manualmente un IME, las siguientes llamadas de ActionScript a métodos y propiedades IME funcionarán como se esperaba. Por ejemplo, si se utiliza un IME japonés, debe estar activado antes de llamar a cualquier método o propiedad del IME.*

Comprobar si un IME está instalado y activado

Antes de llamar a cualquiera de los métodos o propiedades del IME, siempre se debe comprobar si en el equipo del usuario hay actualmente un IME instalado y activado. El código siguiente ilustra la manera de comprobar que el usuario tiene un IME instalado y activado antes de llamar a sus métodos:

```
if (Capabilities.hasIME)
{
    if (IME.enabled)
    {
        trace("IME is installed and enabled.");
    }
    else
    {
        trace("IME is installed but not enabled. Please enable your IME and try again.");
    }
}
else
{
    trace("IME is not installed. Please install an IME and try again.");
}
```

El código anterior comprueba primero si el usuario tiene un IME instalado mediante la propiedad `Capabilities.hasIME`. Si esta propiedad está establecida en `true`, el código consulta la propiedad `IME.enabled` para comprobar si el IME del usuario está activado actualmente.

Determinar el modo de conversión del IME activado actualmente

Al crear aplicaciones multilingües, es posible que sea necesario determinar el modo de conversión que el usuario tiene activo actualmente. El código siguiente ilustra la manera de comprobar si el usuario tiene un IME instalado y, en caso afirmativo, qué modo de conversión de IME está activo actualmente:

```
if (Capabilities.hasIME)
{
    switch (IME.conversionMode)
    {
        case IMEConversionMode.ALPHANUMERIC_FULL:
            tf.text = "Current conversion mode is alphanumeric (full-width).";
            break;
        case IMEConversionMode.ALPHANUMERIC_HALF:
            tf.text = "Current conversion mode is alphanumeric (half-width).";
            break;
        case IMEConversionMode.CHINESE:
            tf.text = "Current conversion mode is Chinese.";
            break;
        case IMEConversionMode.JAPANESE_HIRAGANA:
            tf.text = "Current conversion mode is Japananese Hiragana.";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_FULL:
            tf.text = "Current conversion mode is Japanese Katakana (full-width).";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_HALF:
            tf.text = "Current conversion mode is Japanese Katakana (half-width).";
            break;
        case IMEConversionMode.KOREAN:
            tf.text = "Current conversion mode is Korean.";
            break;
        default:
            tf.text = "Current conversion mode is " + IME.conversionMode + ".";
            break;
    }
}
else
{
    tf.text = "Please install an IME and try again.";
}
```

El código anterior comprueba primero si el usuario tiene un IME instalado. A continuación comprueba el modo de conversión utilizado por el IME actual comparando el valor de la propiedad `IME.conversionMode` con cada una de las constantes de la clase IMEConversionMode.

Definición del modo de conversión del IME

Cuando se cambia el modo de conversión del IME del usuario, es necesario asegurarse de que el código está incluido en un bloque `try..catch`, porque definir un modo de conversión utilizando la propiedad `conversionMode` puede generar un error si el IME no puede establecer el modo de conversión. El código siguiente muestra cómo utilizar un bloque `try..catch` al establecer la propiedad `IME.conversionMode`:

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
    try
    {
        IME.enabled = true;
        IME.conversionMode = IMEConversionMode.KOREAN;
        statusText.text = "Conversion mode is " + IME.conversionMode + ".";
    }
    catch (error:Error)
    {
        statusText.text = "Unable to set conversion mode.\n" + error.message;
    }
}
```

El código anterior crea primero un campo de texto, que se utiliza para mostrar un mensaje de estado al usuario. A continuación, si el IME está instalado, el código lo activa y establece el modo de conversión en Coreano. Si el equipo del usuario no tiene un IME coreano instalado, Flash Player o AIR emite un error que es detectado por el bloque `try..catch`. El bloque `try..catch` muestra el mensaje de error en el campo de texto creado previamente.

Desactivar el IME para determinados campos de texto

En algunos casos, es posible que se desee desactivar el IME del usuario mientras éste escribe caracteres. Por ejemplo, si hay un campo de texto que sólo acepta entradas numéricas, es posible que no interese que aparezca el IME y ralentice la entrada de datos.

En el ejemplo siguiente se ilustra la manera de detectar eventos `FocusEvent.FOCUS_IN` y `FocusEvent.FOCUS_OUT`, y de desactivar el IME del usuario:

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = false;
    }
}
function focusOutHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = true;
    }
}
```

Este ejemplo crea dos campos de entrada de texto, `phoneTxt` y `nameTxt`, y después añade dos detectores de eventos al campo de texto `phoneTxt`. Cuando el usuario establece la selección en el campo de texto `phoneTxt`, se distribuye un evento `FocusEvent.FOCUS_IN` y se desactiva el IME. Cuando el campo de texto `phoneTxt` deja de estar seleccionado, se distribuye el evento `FocusEvent.FOCUS_OUT` para volver a activar el IME.

Detección de eventos de composición del IME

Cuando se establece una cadena de composición, se distribuyen eventos de composición del IME. Por ejemplo, si el usuario activa su IME y escribe una cadena en japonés, se distribuye el evento `IMEEvent.IME_COMPOSITION` en cuanto el usuario seleccione la cadena de composición. Para poder detectar el evento `IMEEvent.IME_COMPOSITION`, hay que añadir un detector de eventos a la propiedad estática `ime` de la clase System (`flash.system.System.ime.addEventListener(...)`), como se indica en el siguiente ejemplo:

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
    IME.enabled = true;
    try
    {
        IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
    }
    catch (error:Error)
    {
        outputTxt.text = "Unable to change IME.";
    }
    System.ime.addEventListener(IMEEvent.IME_COMPOSITION, imeCompositionHandler);
}
else
{
    outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
    outputTxt.text = "you typed: " + event.text;
}
```

El código anterior crea dos campos de texto y los añade a la lista de visualización. El primer campo de texto, `inputTxt`, es un campo de entrada de texto que permite al usuario escribir texto en japonés. El segundo campo de texto, `outputTxt`, es un campo de texto dinámico que muestra mensajes de error al usuario o reproduce la cadena en japonés que el usuario escribe en el campo de texto `inputTxt`.

Ejemplo: Detección de las características del sistema

En el ejemplo CapabilitiesExplorer se muestra la manera de utilizar la clase flash.system.Capabilities para determinar qué funciones admite la versión del reproductor Flash Player o AIR del usuario. Este ejemplo ilustra las técnicas siguientes:

- Detectar las características de la versión del reproductor Flash Player o AIR del usuario mediante la clase Capabilities
- Utilizar la clase ExternalInterface para detectar la configuración del navegador del usuario

Para obtener los archivos de la aplicación para esta muestra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación CapabilitiesExplorer se encuentran en la carpeta Samples/CapabilitiesExplorer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
CapabilitiesExplorer.fla o CapabilitiesExplorer.mxml	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	La clase que proporciona la funcionalidad principal de la aplicación, como añadir las características del sistema a un conjunto, ordenar los elementos y utilizar la clase ExternalInterface para obtener las características del navegador.
capabilities.html	Un contenedor HTML que contiene el código JavaScript necesario para comunicarse con la API externa.

Información general sobre CapabilitiesExplorer

El archivo CapabilitiesExplorer.mxml se encarga de configurar la interfaz de usuario para la aplicación CapabilitiesExplorer. Las características de la versión de Flash Player o AIR del usuario se mostrarán en una instancia del componente DataGrid en el escenario. También se mostrarán las características del navegador si se ejecuta la aplicación desde un contenedor HTML y la API externa está disponible.

Cuando se distribuye el evento `creationComplete` del archivo de aplicación principal, se invoca el método `initApp()`. El método `initApp()` llama al método `getCapabilities()` desde la clase com.example.programmingas3.capabilities.CapabilitiesGrabber. El código del método `initApp()` es el siguiente:

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

El método `CapabilitiesGrabber.getCapabilities()` devuelve un conjunto ordenado con las características de AIR, Flash Player y el navegador, que después se establece en la propiedad `dataProvider` de la instancia `capabilitiesGrid` del componente DataGrid en el objeto Stage.

Información general sobre la clase CapabilitiesGrabber

El método estático `getCapabilities()` de la clase CapabilitiesGrabber añade cada propiedad de la clase flash.system.Capabilities a un conjunto (`capDP`). Después llama al método estático `getBrowserObjects()` de la clase CapabilitiesGrabber. El método `getBrowserObjects()` utiliza la API externa para recorrer el objeto navigator del navegador, que contiene las características del navegador. El código del método `getCapabilities()` es:

```
public static function getCapabilities():Array
{
    var capDP:Array = new Array();
    capDP.push({name:"Capabilities.avHardwareDisable", value:Capabilities.avHardwareDisable});
    capDP.push({name:"Capabilities.hasAccessibility", value:Capabilities.hasAccessibility});
    capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
    ...
    capDP.push({name:"Capabilities.version", value:Capabilities.version});
    var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
    if (navArr.length > 0)
    {
        capDP = capDP.concat(navArr);
    }
    capDP.sortOn("name", Array.CASEINSENSITIVE);
    return capDP;
}
```

El método `getBrowserObjects()` devuelve un conjunto que contiene cada una de las propiedades del objeto navigator del navegador. Si este conjunto tiene una longitud de uno o más elementos, el conjunto de características del navegador (`navArr`) se añade a la matriz de características de Flash Player (`capDP`) y se ordena alfabéticamente el conjunto completo. Por último, el conjunto ordenado se devuelve al archivo de aplicación principal, que llena a continuación la cuadrícula de datos. El código del método `getBrowserObjects()` es el siguiente:

```
private static function getBrowserObjects():Array
{
    var itemArr:Array = new Array();
    var itemVars:URLVariables;
    if (ExternalInterface.available)
    {
        try
        {
            var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
            itemVars = new URLVariables(tempStr);
            for (var i:String in itemVars)
            {
                itemArr.push({name:i, value:itemVars[i]});
            }
        }
        catch (error:SecurityError)
        {
            // ignore
        }
    }
    return itemArr;
}
```

Si la API externa está disponible en el entorno de usuario actual, Flash Player llama al método `JS_getBrowserObjects()` de JavaScript, que recorre el objeto navigator del navegador y devuelve a ActionScript una cadena de valores codificados en URL. Esta cadena se convierte en un objeto URLVariables (`itemVars`) y se añade al conjunto `itemArr`, que se devuelve al script que hizo la llamada.

Comunicación con JavaScript

La parte final de la creación de la aplicación CapabilitiesExplorer consiste en escribir el código JavaScript necesario para recorrer cada uno de los elementos del objeto navigator del navegador y añadir un par nombre-valor a un conjunto temporal. El código del método `JS_getBrowserObjects()` de JavaScript del archivo container.html es:

```
<script language="JavaScript">
    function JS_getBrowserObjects()
    {
        // Create an array to hold each of the browser's items.
        var tempArr = new Array();

        // Loop over each item in the browser's navigator object.
        for (var name in navigator)
        {
            var value = navigator[name];

            // If the current value is a string or Boolean object, add it to the
            // array, otherwise ignore the item.
            switch (typeof(value))
            {
                case "string":
                case "boolean":

                    // Create a temporary string which will be added to the array.
                    // Make sure that we URL-encode the values using JavaScript's
                    // escape() function.
                    var tempStr = "navigator." + name + "=" + escape(value);
                    // Push the URL-encoded name/value pair onto the array.
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Loop over each item in the browser's screen object.
        for (var name in screen)
        {
            var value = screen[name];

            // If the current value is a number, add it to the array, otherwise
            // ignore the item.
            switch (typeof(value))
            {
                case "number":
                    var tempStr = "screen." + name + "=" + escape(value);
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Return the array as a URL-encoded string of name-value pairs.
        return tempArr.join("&");
    }
</script>
```

El código empieza creando un conjunto temporal que contendrá todos los pares nombre-valor del objeto navigator. A continuación, el objeto navigator se somete a un bucle `for..in` y se evalúa el tipo de datos del valor actual para filtrar y omitir los valores no deseados. En esta aplicación, sólo estamos interesados en valores de cadena o booleanos, y los otros tipos de datos (como funciones o conjuntos) se omiten. Cada valor de tipo cadena o booleano del objeto navigator se añade al conjunto `tempArr`. A continuación, el objeto screen del navegador se somete a un bucle `for..in` y se añaden los valores numéricos encontrados al conjunto `tempArr`. Por último, el conjunto se convierte en una cadena mediante el método `Array.join()`. El conjunto usa un carácter ampersand (&) como delimitador, lo que permite a ActionScript analizar fácilmente los datos con la clase URLVariables.

Capítulo 29: Copiar y pegar

Utilice las clases de la API de portapapeles para copiar información en el portapapeles del sistema. Entre los formatos de datos que se pueden transferir entre una aplicación que se ejecuta en Adobe® AIR™ y Adobe® Flash® Player se incluyen:

- Mapas de bits (sólo AIR)
- Archivos (sólo AIR)
- Texto
- Texto en formato HTML
- Datos con formato de texto enriquecido
- Cadenas URL (sólo AIR)
- Objetos serializados
- Referencias a objetos (sólo válido en la aplicación original)

Conceptos básicos de copiar y pegar

La API de copiar y pegar contiene las siguientes clases.

Paquete	Clases
flash.desktop	• Clipboard • ClipboardFormats • ClipboardTransferMode

La propiedad estática `Clipboard.generalClipboard` representa el portapapeles del sistema operativo. La clase Clipboard proporciona métodos para leer y escribir datos en objetos Clipboard.

La clases HTMLLoader (en AIR) y TextField implementan el comportamiento predeterminado para los métodos abreviados de teclado normales de copiar y pegar. Para implementar el comportamiento del método abreviado de copiar y pegar para componentes personalizados, puede detectar estas pulsaciones de teclas directamente. También puede utilizar comandos de menú nativos junto con equivalentes de teclas para responder a las pulsaciones de teclas indirectamente.

Se puede disponer de distintas representaciones de la misma información en un solo objetos Clipboard para aumentar las posibilidades de que otras aplicaciones puedan entender y utilizar los datos. Por ejemplo, una imagen puede incluirse como datos de imagen, como un objeto Bitmap serializado y como un archivo. Se puede retrasar la representación de los datos en un formato determinado para no tener que crear el formato hasta que se lean los datos.

Lectura y escritura en el portapapeles del sistema

Para leer el portapapeles del sistema operativo, llame al método `getData()` del objeto `Clipboard.generalClipbooard` y transfiera el nombre del formato que se quiere leer:

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

Nota: *el contenido que se ejecuta en Flash Player o en un entorno limitado que no pertenezca a la aplicación en AIR sólo puede llamar al método* `getData()` *en un controlador de eventos para un evento* `paste`*. Únicamente el código que se ejecuta en el entorno limitado de la aplicación de AIR puede llamar al método* `getData()` *fuera de un controlador de eventos* `paste`*.*

Para escribir en el portapapeles, añada los datos al objeto `Clipboard.generalClipboard` en uno o varios formatos. Todos los datos existentes con el mismo formato se sobrescribirán automáticamente. No obstante, se recomienda borrar también el portapapeles del sistema antes de escribir datos nuevos para asegurarse de que también se borran todos los datos no relacionados en otros formatos.

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

var textToCopy:String = "Copy to clipboard.";
Clipboard.generalClipboard.clear();
Clipboard.generalClipboard.setData(ClipboardFormats.TEXT_FORMAT, textToCopy, false);
```

Nota: *el contenido que se ejecuta en Flash Player o en un entorno limitado que no pertenezca a la aplicación en AIR sólo puede llamar al método* `setData()` *en un controlador de eventos de usuario, por ejemplo, eventos de teclado o de ratón, o de eventos* `copy` *o* `cut`*. Sólo el código que se ejecuta en el entorno limitado de la aplicación de AIR puede llamar al método* `setData()` *desde fuera de un controlador de eventos de usuario.*

Formatos de datos del portapapeles

Los formatos del portapapeles describen los datos colocados en un objeto Clipboard. Flash Player o AIR convierten automáticamente los formatos de datos estándar entre tipos de datos de ActionScript y formatos del portapapeles del sistema. Además, los objetos de la aplicación se pueden transferir entre aplicaciones basadas en ActionScript mediante formatos definidos por la aplicación.

Un objeto Clipboard puede contener representaciones de la misma información con distintos formatos. Por ejemplo, un objeto Clipboard que representa a un elemento Sprite puede incluir un formato de referencia para utilizarse dentro de la misma aplicación, un formato serializado para utilizarlo con otra aplicación que se ejecute en Flash Player o AIR, un formato de mapa de bits para utilizarse en un editor de imágenes y un formato de lista de archivos (posiblemente con representación aplazada para codificar un archivo PNG) para copiar o arrastrar una representación del elemento Sprite al sistema de archivos.

Formatos de datos estándar

Las constantes que definen los nombres de formatos estándar se incluyen en la clase ClipboardFormats:

Constante	Descripción
TEXT_FORMAT	Los datos con formato de texto se convierten con la clase String de ActionScript.
HTML_FORMAT	Texto con marcas HTML.
RICH_TEXT_FORMAT	Los datos con formato de texto enriquecido se convierten con la clase ByteArray de ActionScript. El marcado RTF no se interpreta ni se convierte.
BITMAP_FORMAT	(Sólo AIR) Los datos con formato de mapa de bits se convierten con la clase BitmapData de ActionScript.
FILE_LIST_FORMAT	(Sólo AIR) Los datos con formato de lista de archivos se convierten en un conjunto de objetos File de ActionScript.
URL_FORMAT	(Sólo AIR) Los datos con formato URL se convierten con la clase String de ActionScript.

Formatos de datos personalizados

Es posible utilizar formatos personalizados definidos por la aplicación para transferir objetos como referencias o como copias serializadas. Las referencias sólo son válidas en la misma aplicación que se ejecuta en AIR o Flash Player. Los objetos serializados se pueden transferir entre aplicaciones que se ejecutan en AIR o Flash Player, pero sólo se pueden utilizar con objetos que no pierdan su validez al serializarse o deserializarse. Normalmente, los objetos se pueden serializar si sus propiedades son tipos simples u objetos serializables.

Para añadir un objeto serializado a un objeto Clipboard, establezca el parámetros serializable como `true` cuando llame al método `Clipboard.setData()`. El nombre del formato puede ser uno de los nombres estándar o una cadena arbitraria definida por la aplicación.

Modos de transferencia

Cuando un objeto se escribe en el portapapeles con un formato de datos personalizado, los datos del objeto se pueden leer en el portapapeles, bien como una referencia o como una copia serializada del objeto original. AIR define cuatro modos de transferencia que determinan si los objetos se transfieren como referencias o como copias serializadas:

Modo de transferencia	Descripción
ClipboardTransferModes.ORIGINAL_ONLY	Sólo se devuelve una referencia. Si no hay ninguna disponible, se devuelve un valor null.
ClipboardTransferModes.ORIGINAL_PREFFERED	Se devuelve una referencia, si está disponible. En caso contrario, se devuelve una copia serializada.
ClipboardTransferModes.CLONE_ONLY	Sólo se devuelve una copia serializada. Si no hay ninguna disponible, se devuelve un valor null.
ClipboardTransferModes.CLONE_PREFFERED	Se devuelve una copia serializada, si está disponible. En caso contrario, se devuelve una referencia.

Lectura y escritura de formatos de datos personalizados

Se puede utilizar cualquier cadena que no empiece con los prefijos reservados `air:` o `flash:` en el parámetro de formato al escribir un objeto en el portapapeles. Se recomienda utilizar la misma cadena que la del formato para leer el objeto. En el siguiente ejemplo se muestra cómo leer y escribir objetos en el portapapeles:

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}
```

Para extraer un objeto serializado del objeto Clipboard (tras una operación de soltar y pegar), utilice el mismo nombre de formato y los modos de transferencia `cloneOnly` o `clonePreferred`.

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

Siempre se añade una referencia al objeto Clipboard. Para extraer la referencia del objeto Clipboard (tras una operación de soltar y pegar), en lugar de utilizar la copia serializada, utilice los modos de transferencia `originalOnly` o `originalPreferred`:

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

Las referencias sólo son válidas si el objeto Clipboard se origina en la aplicación actual que se ejecuta en AIR o Flash Player. Utilice el modo de transferencia `originalPreferred` para acceder a la referencia si está disponible, y al clon serializado si no está disponible.

Representación aplazada

Si crear un formato de datos requiere mucho trabajo de programación, es posible utilizar la representación aplazada suministrando una función que proporcione los datos cuando se necesiten. Sólo se llamará a esta función si un receptor de la operación de soltar y pegar requiere los datos en formato aplazado.

La función de representación se añade a un objeto Clipboard mediante el método `setDataHandler()`. La función debe devolver los datos en el formato correcto. Por ejemplo, si llama a `setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText)`, la función `writeText()` debe devolver una cadena.

Si se añade un formato de datos del mismo tipo a un objeto Clipboard con el método `setData()`, dichos datos tienen prioridad sobre la versión aplazada (y no se llamará nunca a la función de representación). Se puede volver a llamar (o no) a la función de representación si se vuelve a acceder a los mismos datos del portapapeles una segunda vez.

Nota: *en Mac OS X, la representación aplazada funciona sólo con formatos de datos personalizados. En los formatos de datos estándar, se llama a la función de representación inmediatamente.*

Pegar texto mediante una función de representación aplazada

En el siguiente ejemplo se muestra cómo implementar una función de representación aplazada.

Cuando el usuario pulsa el botón Copiar, la aplicación borra el portapapeles del sistema para garantizar que no queda contenido de operaciones anteriores con el portapapeles. El método `setDataHandler()` establece entonces la función `renderData()` como procesador del portapapeles.

Cuando el usuario selecciona el comando Pegar en el menú contextual del campo de texto de destino, la aplicación accede al portapapeles y establece el texto de destino. Dado que el formato de los datos de texto del portapapeles se ha establecido con una función y no con una cadena, el portapapeles llama a la función `renderData()`. La función `renderData()` devuelve el texto de origen que, después, se asigna al texto de destino.

Es preciso tener en cuenta que si se modifica el texto de origen antes de pulsar el botón Pegar, los cambios se reflejarán en el texto pegado, incluso si la modificación se produce después de haber pulsado el botón Copiar. Esto se debe a que la función de representación no copia el texto de origen hasta que no se pulsa el botón Pegar. (Al utilizar la representación aplazada en una aplicación real, se deben almacenar o proteger los datos de origen de algún modo para evitar este problema.)

```
package {
   import flash.desktop.Clipboard;
   import flash.desktop.ClipboardFormats;
   import flash.desktop.ClipboardTransferMode;
   import flash.display.Sprite;
   import flash.text.TextField;
   import flash.text.TextFormat;
   import flash.text.TextFieldType;
   import flash.events.MouseEvent;
   import flash.events.Event;
   public class DeferredRenderingExample extends Sprite
   {
       private var sourceTextField:TextField;
       private var destination:TextField;
       private var copyText:TextField;
       public function DeferredRenderingExample():void
       {
           sourceTextField = createTextField(10, 10, 380, 90);
           sourceTextField.text = "Neque porro quisquam est qui dolorem "
               + "ipsum quia dolor sit amet, consectetur, adipisci velit.";

           copyText = createTextField(10, 110, 35, 20);
           copyText.htmlText = "<a href='#'>Copy</a>";
           copyText.addEventListener(MouseEvent.CLICK, onCopy);

           destination = createTextField(10, 145, 380, 90);
           destination.addEventListener(Event.PASTE, onPaste);
       }
       private function createTextField(x:Number, y:Number, width:Number,
                       height:Number):TextField
       {
           var newTxt:TextField = new TextField();
           newTxt.x = x;
           newTxt.y = y;
           newTxt.height = height;
           newTxt.width = width;
           newTxt.border = true;
           newTxt.multiline = true;
           newTxt.wordWrap = true;
           newTxt.type = TextFieldType.INPUT;
           addChild(newTxt);
           return newTxt;
       }
       public function onCopy(event:MouseEvent):void
       {
           Clipboard.generalClipboard.clear();
           Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
                           renderData);
       }
       public function onPaste(event:Event):void
       {
```

```
            sourceTextField.text =
            Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
        }
        public function renderData():String
        {
            trace("Rendering data");
            var sourceStr:String = sourceTextField.text;
            if (sourceTextField.selectionEndIndex >
                     sourceTextField.selectionBeginIndex)
            {
                return sourceStr.substring(sourceTextField.selectionBeginIndex,
                                    sourceTextField.selectionEndIndex);
            }
            else
            {
                return sourceStr;
            }
        }
    }
}
```

Capítulo 30: Impresión

Adobe® Flash® Player y Adobe® AIR™ pueden comunicarse con la interfaz de impresión de un sistema operativo para poder pasar páginas a la cola de impresión. Cada página que Flash Player o AIR envían a la cola puede incluir contenido visible, dinámico o fuera de la pantalla para el usuario, incluidos valores de base de datos y texto dinámico. Asimismo, Flash Player y AIR establecen las propiedades de la clase flash.printing.PrintJob en función de la configuración de impresora del usuario, por lo que puede aplicar un formato adecuado a las páginas.

En este capítulo se detallan las estrategias para utilizar los métodos y las propiedades de la clase flash.printing.PrintJob para crear un trabajo de impresión, leer la configuración de impresión de un usuario y realizar ajustes en un trabajo de impresión basándose en la respuesta de Flash Player o AIR y del sistema operativo del usuario.

Fundamentos de impresión

Introducción a la impresión

En ActionScript 3.0 se utiliza la clase PrintJob para crear instantáneas de contenido de la pantalla con el fin de convertirlas en las representaciones en tinta y papel de una copia impresa. En cierto modo, configurar contenido para imprimir es lo mismo que configurarlo para la visualización en pantalla: se disponen los elementos y se ajusta su tamaño para crear el diseño deseado. Sin embargo, la impresión tiene peculiaridades que la distinguen de la visualización en pantalla. Por ejemplo, la resolución de las impresoras es distinta de la de los monitores de ordenador, el contenido de una pantalla de ordenador es dinámico y puede cambiar mientras que el contenido impreso es estático, y al planificar la impresión hay que tener en cuenta las restricciones de un tamaño fijo de página y la posibilidad de la impresión multipágina.

Aunque estas diferencias pueden parecer obvias, es importante tenerlas en cuenta al configurar la impresión con ActionScript. Como la precisión de la impresión depende de una combinación de los valores especificados por el programador y las características de la impresora del usuario, la clase PrintJob incluye propiedades que permiten determinar las características importantes de la impresora del usuario que hay que tener en cuenta.

Tareas comunes de impresión

En este capítulo se describen las siguientes tareas comunes relacionadas con la impresión:

- Iniciar un trabajo de impresión
- Añadir páginas a un trabajo de impresión
- Determinar si el usuario cancela un trabajo de impresión
- Especificar si se debe utilizar representación de mapas de bits o de vectores
- Configurar el tamaño, la escala y la orientación
- Especificar el área de contenido imprimible
- Convertir el tamaño de pantalla al tamaño de página
- Imprimir trabajos de impresión multipágina

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Cola de impresión: parte del sistema operativo o software controlador de impresora que hace un seguimiento de las páginas en espera de ser impresas y las envía a la impresora cuando ésta disponible.
- Orientación de la página: rotación del contenido impreso con relación al papel (horizontal o vertical).
- Trabajo de impresión: página o conjunto de páginas que constituyen una impresión individual.

Ejecución de los ejemplos del capítulo

A medida que progrese en el estudio de este capítulo, podría desear probar los listados de código de ejemplo. Muchos de los listados de código del capítulo son pequeños fragmentos de código, no ejemplos completos de impresión o código de comprobación de valores. Para probar los ejemplos hay que crear los elementos que se van a imprimir y utilizar los listados de código con esos elementos. Los dos últimos ejemplos del capítulo son ejemplos completos de tareas de impresión que incluyen el código que define el contenido que se va a imprimir y lleva a cabo las tareas de impresión.

Para probar los listados de código de ejemplo:

1. Cree un documento de Flash nuevo.
2. Seleccione el fotograma clave del Fotograma 1 de la línea de tiempo y abra el panel Acciones.
3. Copie el listado de código en el panel Script.
4. En el menú principal, elija Control > Probar película para crear el archivo SWF y probar el ejemplo.

Impresión de una página

Para controlar la impresión se puede utilizar una instancia de la clase PrintJob. Para imprimir una página básica a través de Flash Player o AIR, se debe utilizar la siguiente secuencia de cuatro sentencias:

- `new PrintJob()`: crea una nueva instancia de trabajo de impresión con el nombre especificado por el usuario.
- `PrintJob.start()`: inicia el proceso de impresión en el sistema operativo, llama al cuadro de diálogo de impresión para el usuario y llena los valores de las propiedades de sólo lectura del trabajo de impresión.
- `PrintJob.addPage()`: contiene los detalles relativos al contenido del trabajo de impresión, como el objeto Sprite (y sus elementos secundarios), el tamaño del área de impresión y si la impresora debe imprimir la imagen como un vector o un mapa de bits. Se pueden utilizar llamadas sucesivas a `addPage()` para imprimir varios objetos Sprite en varias páginas.
- `PrintJob.send()`: envía las páginas a la impresora del sistema operativo.

Así pues, por ejemplo, un script de trabajo de impresión muy sencillo tendría el siguiente aspecto (con las sentencias de compilación `package`, `import` y `class`):

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;

    public class BasicPrintExample extends Sprite
    {
        var myPrintJob:PrintJob = new PrintJob();
        var mySprite:Sprite = new Sprite();

        public function BasicPrintExample()
        {
            myPrintJob.start();
            myPrintJob.addPage(mySprite);
            myPrintJob.send();
        }
    }
}
```

Nota: *este ejemplo pretende mostrar los elementos básicos de un script de trabajo de impresión y no contiene gestión de errores. Para generar un script que responda adecuadamente a la cancelación de un trabajo de impresión por parte del usuario, consulte "Trabajo con excepciones y valores devueltos" en la página 686.*

Si es necesario borrar las propiedades de un objeto PrintJob por algún motivo, debe establecerse la variable PrintJob en `null` (por ejemplo, `myPrintJob = null`).

Interfaz de impresión del sistema y tareas de Flash Player y AIR

Debido a que Flash Player y AIR distribuyen páginas a la interfaz del impresión del sistema operativo, deben conocerse las tareas administradas por ambas aplicaciones y las tareas administradas por la interfaz de impresión del sistema operativo. Flash Player y AIR pueden iniciar un trabajo de impresión, leer parte de la configuración de página de una impresora, pasar el contenido de un trabajo de impresión al sistema operativo y verificar si el usuario o el sistema han cancelado un trabajo de impresión. Otros procesos, como mostrar los cuadros de diálogo específicos de la impresora, cancelar un trabajo de impresión en cola o notificar el estado de la impresora son tareas que gestiona el sistema operativo. Flash Player y AIR pueden responder si hay un problema para iniciar o dar formato a un trabajo de impresión, pero sólo pueden ofrecer información sobre determinadas propiedades o condiciones de la interfaz de usuario del sistema operativo. Como desarrollador, su código debe poder responder a estas propiedades o condiciones.

Trabajo con excepciones y valores devueltos

Si el usuario ha cancelado el trabajo de impresión, hay que comprobar si el método `PrintJob.start()` devuelve `true` antes de ejecutar llamadas a `addPage()` y `send()`. Una forma sencilla de comprobar si se han cancelado estos métodos antes de continuar es incluirlos en una sentencia `if`, como en el siguiente ejemplo:

```
if (myPrintJob.start())
{
    // addPage() and send() statements here
}
```

Si el valor de `PrintJob.start()` es `true`, lo que significa que el usuario ha seleccionado Imprimir (o que Flash Player o AIR han iniciado un comando Print), se puede llamar a los métodos `addPage()` y `send()`.

Asimismo, para ayudar a administrar el proceso de impresión, Flash Player y Adobe AIR emiten excepciones para el método `PrintJob.addPage()`, de forma que es posible capturar errores y proporcionar información y opciones al usuario. Si un método `PrintJob.addPage()` no se ejecuta correctamente, se puede llamar a otra función o detener el trabajo de impresión actual. Estas excepciones se pueden capturar incorporando llamadas `addPage()` en una sentencia `try..catch`, tal y como se muestra en el siguiente ejemplo. En el ejemplo, `[params]` es un marcador de posición para los parámetros, que especifica el contenido real que se desea imprimir:

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error,
    }
    myPrintJob.send();
}
```

Cuando se inicia el trabajo de impresión, se puede añadir contenido mediante `PrintJob.addPage()` y ver si se genera una excepción (por ejemplo, si el usuario ha cancelado el trabajo de impresión). Si es así, se puede añadir lógica a la sentencia `catch` para proporcionar al usuario (o a Flash Player o AIR) información y opciones, o bien, se puede detener el trabajo de impresión actual. Si se añade la página correctamente, se puede continuar enviando las páginas a la impresora mediante `PrintJob.send()`.

Si Flash Player o AIR encuentran un problema al enviar el trabajo de impresión a la impresora (por ejemplo, si la impresora está sin conexión), se puede capturar también dicha excepción y proporcionar al usuario (o a Flash Player o AIR) información o más opciones (por ejemplo, mostrar el texto de un mensaje o proporcionar una alerta en una animación). Por ejemplo, puede asignar texto nuevo a un campo de texto en una sentencia `if..else`, tal y como se indica en el siguiente código:

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error.
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

Para ver ejemplos de uso, consulte “Ejemplo: Ajuste de escala, recorte y respuesta” en la página 692.

Trabajo con las propiedades de página

Cuando el usuario hace clic en Aceptar en el cuadro de diálogo Imprimir y `PrintJob.start()` devuelve `true`, se puede acceder a las propiedades definidas por la configuración de la impresora. Estas propiedades se refieren a la anchura del papel, la altura del papel (`pageWidth` y `pageHeight`) y a la orientación del contenido en el papel. Debido a que se trata de la configuración de impresora y Flash Player y AIR no la controlan, no es posible modificarla; sin embargo, se puede utilizar para alinear el contenido enviado a la impresora para que coincida con la configuración actual. Para obtener más información, consulte "Configuración del tamaño, la escala y la orientación" en la página 689.

Configuración de la representación vectorial o de mapa de bits

Es posible configurar manualmente el trabajo de impresión para que cada una de las páginas se coloque en cola como un gráfico vectorial o una imagen de mapa de bits. En algunos casos, la impresión vectorial producirá un archivo de cola más pequeño y una imagen mejor que la impresión de mapa de bits. Sin embargo, si el contenido incluye una imagen de mapa de bits y se desea conservar la transparencia alfa o los efectos de color, se debe imprimir la página como una imagen de mapa de bits. Además, las impresoras que no son PostScript convierten automáticamente los gráficos vectoriales en imágenes de mapa de bits.

Nota: *Adobe AIR no admite la impresión vectorial en Mac OS.*

La impresión de mapa de bits se especifica en el tercer parámetro de `PrintJob.addPage()`, pasando un objeto PrintJobOptions con el parámetro `printAsBitmap` establecido en `true`, como en el siguiente ejemplo:

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

Si no se especifica un valor para el tercer parámetro, el trabajo de impresión utilizará el valor predeterminado, que es la impresión vectorial.

Nota: *si no se desea especificar un valor para `printArea` (el segundo parámetro) pero sí para la impresión de mapa de bits, hay que utilizar `null` para `printArea`.*

Sincronización de las sentencias del trabajo de impresión

ActionScript 3.0 no restringe un objeto PrintJob a un solo fotograma (como ocurría en versiones anteriores de ActionScript). Sin embargo, como el sistema operativo muestra al usuario información del estado de la impresión después de que el usuario haya hecho clic en el botón Aceptar del cuadro de diálogo Imprimir, debería llamarse a `PrintJob.addPage()` y `PrintJob.send()`, en cuanto sea posible, para enviar páginas a la cola. Una demora al llegar al fotograma que contiene la llamada a `PrintJob.send()` retrasará el proceso de impresión.

En ActionScript 3.0 hay un límite de tiempo de espera de script de 15 segundos. Por lo tanto, el tiempo entre cada sentencia principal de una secuencia de un trabajo de impresión no puede superar los 15 segundos. Dicho de otro modo, el límite de tiempo de espera del script de 15 segundos se aplica a los siguientes intervalos:

- Entre `PrintJob.start()` y el primer `PrintJob.addPage()`
- Entre `PrintJob.addPage()` y el siguiente `PrintJob.addPage()`
- Entre el último `PrintJob.addPage()` y `PrintJob.send()`

Si alguno de los intervalos anteriores dura más de 15 segundos, la siguiente llamada a `PrintJob.start()` en la instancia de PrintJob devolverá `false` y el siguiente `PrintJob.addPage()` de la instancia de PrintJob hará que Flash Player o AIR emitan una excepción de tiempo de ejecución.

Configuración del tamaño, la escala y la orientación

En la sección "Impresión de una página" en la página 685 se detallan los pasos de un trabajo de impresión básico, en el que la salida refleja directamente el equivalente impreso del tamaño de pantalla y la posición del objeto Sprite especificado. Sin embargo, las impresoras utilizan resoluciones distintas para imprimir y pueden tener configuraciones que afecten negativamente al aspecto del objeto Sprite impreso.

Flash Player y AIR pueden leer la configuración de impresión de un sistema operativo, pero se debe tener en cuenta que estas propiedades son de sólo lectura: aunque se puede responder a sus valores, éstos no pueden definirse. Así pues, por ejemplo, se puede buscar la configuración de tamaño de página de la impresora y ajustar el contenido a dicho tamaño. También se puede determinar la configuración de márgenes y la orientación de página de una impresora. Para responder a la configuración de la impresora, es posible que sea necesario especificar un área de impresión, ajustar la diferencia entre la resolución de una pantalla y las medidas de puntos de una impresora, o transformar el contenido para ajustarlo a la configuración de tamaño u orientación de la impresora del usuario.

Utilización de rectángulos en el área de impresión

El método `PrintJob.addPage()` permite especificar la región de un objeto Sprite que se desea imprimir. El segundo parámetro, `printArea` tiene la forma de un objeto Rectangle. Hay tres formas posibles de proporcionar un valor para este parámetro:

- Crear un objeto Rectangle con propiedades específicas y luego utilizar el rectángulo en la llamada a `addPage()`, como en el siguiente ejemplo:

```
private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
myPrintJob.addPage(sheet, rect1);
```

- Si no se ha especificado todavía un objeto Rectangle, puede hacerse en la propia llamada, como en el siguiente ejemplo:

```
myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
```

- Si se ha previsto proporcionar valores para el tercer parámetro en la llamada a `addPage()`, pero no se desea especificar un rectángulo, se puede utilizar `null` para el segundo parámetro, como en el siguiente ejemplo:

```
myPrintJob.addPage(sheet, null, options);
```

Nota: *si se ha previsto especificar un rectángulo para las dimensiones de impresión, es necesario importar la clase* `flash.display.Rectangle`.

Comparación de puntos y píxeles

La anchura y la altura de un rectángulo son valores expresados en píxeles. Una impresora utiliza los puntos como unidad de medida de impresión. Los puntos tienen un tamaño físico fijo (1/72 pulgadas), pero el tamaño de un píxel en pantalla depende de la resolución de cada pantalla. Así, la relación de conversión entre píxeles y puntos depende de la configuración de la impresora y del hecho de que el objeto Sprite tenga ajustada la escala. Un objeto Sprite de 72 píxeles de ancho y sin ajuste de escala se imprimirá con una anchura de una pulgada, y cada punto equivaldrá a un píxel, independientemente de la resolución de la pantalla.

Puede utilizar las equivalencias siguientes para convertir los valores en pulgadas o centímetros en twips o puntos (un twip es 1/20 de un punto):

- 1 punto = 1/72 pulgadas =20 twips
- 1 pulgada = 72 puntos = 1440 twips
- 1 centímetro = 567 twips

Si se omite o pasa de forma incorrecta el parámetro `printArea`, se imprime el área completa del objeto Sprite.

Escala

Si desea realizar un ajuste de escala en un objeto Sprite antes de imprimirlo, establezca las propiedades de escala (consulte "Manipulación del tamaño y ajuste de escala de los objetos" en la página 306) antes de llamar al método `PrintJob.addPage()` y restablezca los valores originales después de imprimir. La escala de un objeto Sprite no tiene relación con la propiedad `printArea`. En otras palabras, si se especifica un área de impresión de 50 por 50 píxeles, se imprimen 2500 píxeles. Si se cambia la escala del objeto Sprite, se seguirán imprimiendo 2500 píxeles, pero el objeto Sprite se imprimirá con el tamaño con la escala ajustada.

Para ver un ejemplo, consulte "Ejemplo: Ajuste de escala, recorte y respuesta" en la página 692.

Impresión de la orientación horizontal o vertical

Flash Player y AIR pueden detectar la configuración de la orientación, por lo que se puede generar lógica en el código ActionScript para ajustar el tamaño del contenido o la rotación como respuesta a la configuración de la impresora, como se muestra en el siguiente ejemplo:

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

Nota: *si va a leer la configuración del sistema para la orientación de contenido en papel, recuerde importar la clase PrintJobOrientation. La clase PrintJobOrientation proporciona valores constantes que definen la orientación del contenido en la página. La case se importa utilizando la siguiente instrucción:*

```
import flash.printing.PrintJobOrientation;
```

Respuesta a la altura y anchura de la página

A través de una estrategia similar a la gestión de la configuración de orientación de la impresora, se puede leer la configuración de altura y anchura de la página, y responder a sus valores mediante la incorporación de lógica en una sentencia `if`. El siguiente código muestra un ejemplo:

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

Además, la configuración de márgenes de una página puede determinarse comparando las dimensiones de la página y del papel, como se muestra en el siguiente ejemplo:

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

Ejemplo: Impresión de varias páginas

Cuando se imprime más de una página de contenido, se puede asociar cada una de las páginas a un objeto Sprite distinto (en este caso, `sheet1` y `sheet2`) y luego utilizar `PrintJob.addPage()` en cada uno de los objetos Sprite. El siguiente código muestra esta técnica:

```
package
{
    import flash.display.MovieClip;
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.geom.Rectangle;

    public class PrintMultiplePages extends MovieClip
    {
        private var sheet1:Sprite;
        private var sheet2:Sprite;

        public function PrintMultiplePages():void
        {
            init();
            printPages();
        }

        private function init():void
        {
            sheet1 = new Sprite();
            createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80, height:130});
            sheet2 = new Sprite();
            createSheet(sheet2, "There was a great story to tell, and it ended quickly.\n\nThe
end.", null);
        }

        private function createSheet(sheet:Sprite, str:String, imgValue:Object):void
        {
            sheet.graphics.beginFill(0xEEEEEE);
            sheet.graphics.lineStyle(1, 0x000000);
            sheet.graphics.drawRect(0, 0, 100, 200);
            sheet.graphics.endFill();

            var txt:TextField = new TextField();
            txt.height = 200;
            txt.width = 100;
            txt.wordWrap = true;
            txt.text = str;

            if (imgValue != null)
            {
                var img:Sprite = new Sprite();
                img.graphics.beginFill(0xFFFFFF);
                img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width, imgValue.height);
                img.graphics.endFill();
                sheet.addChild(img);
            }
            sheet.addChild(txt);
        }

        private function printPages():void
        {
            var pj:PrintJob = new PrintJob();
```

```
            var pagesToPrint:uint = 0;
            if (pj.start())
            {
                if (pj.orientation == PrintJobOrientation.LANDSCAPE)
                {
                    throw new Error("Page is not set to an orientation of portrait.");
                }

                sheet1.height = pj.pageHeight;
                sheet1.width = pj.pageWidth;
                sheet2.height = pj.pageHeight;
                sheet2.width = pj.pageWidth;

                try
                {
                    pj.addPage(sheet1);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                // Respond to error.
                }

                try
                {
                    pj.addPage(sheet2);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                    // Respond to error.
                }

                if (pagesToPrint > 0)
                {
                    pj.send();
                }
            }
        }
    }
}
```

Ejemplo: Ajuste de escala, recorte y respuesta

En algunos casos, es posible que se desee ajustar el tamaño (u otras propiedades) de un objeto de visualización cuando se imprime, con el fin de ajustar las diferencias entre su aspecto en la pantalla y su aspecto impreso en papel. Cuando se ajustan las propiedades de un objeto de visualización antes de imprimir (por ejemplo, usando las propiedades `scaleX` y `scaleY`), hay que tener en cuenta que si se ajusta la escala del objeto más allá del rectángulo definido en el área de impresión, el objeto quedará recortado. También es probable que se deseen restablecer las propiedades después de imprimir las páginas.

El siguiente código ajusta la escala de las dimensiones del objeto de visualización `txt` (pero no el fondo del cuadro verde) y el campo de texto queda recortado por las dimensiones del rectángulo especificado. Después de imprimir, el campo de texto recupera su tamaño original de visualización en la pantalla. Si el usuario cancela el trabajo de impresión desde el cuadro Imprimir del sistema operativo, el contenido de Flash Player o AIR cambia para avisar al usuario de que se ha cancelado el trabajo.

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintScaleExample extends Sprite
    {
        private var bg:Sprite;
        private var txt:TextField;

        public function PrintScaleExample():void
        {
            init();
            draw();
            printPage();
        }

        private function printPage():void
        {
            var pj:PrintJob = new PrintJob();
            txt.scaleX = 3;
            txt.scaleY = 2;
            if (pj.start())
            {
                trace(">> pj.orientation: " + pj.orientation);
                trace(">> pj.pageWidth: " + pj.pageWidth);
                trace(">> pj.pageHeight: " + pj.pageHeight);
                trace(">> pj.paperWidth: " + pj.paperWidth);
                trace(">> pj.paperHeight: " + pj.paperHeight);

                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job canceled";
            }
            // Reset the txt scale properties.
            txt.scaleX = 1;
            txt.scaleY = 1;
```

```
        }

        private function init():void
        {
            bg = new Sprite();
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();

            txt = new TextField();
            txt.border = true;
            txt.text = "Hello World";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

Capítulo 31: Utilización de la API externa

La API externa de ActionScript 3.0 permite disponer de un mecanismo de comunicación sencillo entre ActionScript y la aplicación contenedora dentro de la cual se ejecuta Adobe Flash Player. Existen varias situaciones en las que puede resultar práctico utilizar la API externa como, por ejemplo, al crear una interacción entre un documento SWF y JavaScript en una página HTML, o al desarrollar una aplicación de escritorio que utilice Flash Player para mostrar un archivo SWF.

En este capítulo se explica cómo utilizar la API externa para interactuar con una aplicación contenedora, cómo pasar datos entre ActionScript y JavaScript en una página HTML y cómo establecer una comunicación e intercambiar datos entre ActionScript y una aplicación de escritorio.

Nota: *este capítulo sólo describe la comunicación entre el código ActionScript de un archivo SWF y la aplicación contenedora que incluye una referencia a la instancia de Flash Player en la que se carga el archivo SWF. En esta documentación no se tratan otros usos de Flash Player en una aplicación. Flash Player se ha diseñado para utilizarse como un complemento de navegador o un proyector (aplicación autónoma). Puede haber compatibilidad limitada con otros escenarios de uso.*

Fundamentos de la utilización de la API externa

Introducción a la utilización de la API externa

Si bien en algunos casos un archivo SWF se puede ejecutar de forma independiente (por ejemplo, cuando se crea un proyector SWF), en la mayor parte de las ocasiones las aplicaciones SWF se ejecutan como elementos incorporados dentro de otras aplicaciones. Normalmente, el contenedor que incluye el archivo SWF es un archivo HTML. Menos frecuente es usar un archivo SWF de modo que constituya total o parcialmente la interfaz de usuario de una aplicación de escritorio.

Cuando se trabaja en aplicaciones más avanzadas es posible que sea necesario establecer una comunicación entre el archivo SWF y la aplicación contenedora. Por ejemplo, es habitual que una página Web muestre texto u otra información en HTML e incluya un archivo SWF en el que aparezca contenido visual dinámico, como un diagrama o un vídeo. En esos casos podría resultar útil que cuando los usuarios hicieran clic en un botón de la página Web, algo cambiase en el archivo SWF. ActionScript contiene un mecanismo, conocido como la API externa, que facilita este tipo de comunicación entre el código ActionScript de un archivo SWF y el código de la aplicación contenedora.

Tareas comunes de la API externa

En este capítulo se explican las siguientes tareas comunes de la API externa:

- Obtener información acerca de la aplicación contenedora
- Utilizar ActionScript para llamar a código de una aplicación contenedora, incluida una página Web o una aplicación de escritorio
- Llamar a código de ActionScript desde el código de una aplicación contenedora
- Crear un proxy para simplificar las llamadas al código de ActionScript desde una aplicación contenedora

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en este capítulo:

- Contenedor ActiveX: aplicación contenedora (no un navegador Web) que incluye una instancia del control ActiveX de Flash Player para mostrar contenido SWF en la aplicación.
- Aplicación contenedora: aplicación en la que Flash Player ejecuta un archivo SWF, como un navegador Web y una página HTML que incluye contenido de Flash Player.
- Proyector: archivo SWF que se ha convertido en un ejecutable autónomo en el que se incluye tanto Flash Player como el contenido del archivo SWF. Los proyectores se pueden crear en Adobe Flash CS4 Professional o mediante la versión autónoma de Flash Player. Los proyectores suelen usarse para distribuir archivos SWF en CD-ROM o en otras situaciones similares en las que el tamaño de la descarga no es importante y el autor del SWF desea estar seguro de que los usuarios podrán ejecutar el archivo independientemente de que tengan instalado Flash Player en el equipo.
- Proxy: aplicación o código intermedio que llama a código de una aplicación (la "aplicación externa") en nombre de otra aplicación (la "aplicación que llama") y devuelve valores a esta última. Un proxy se puede usar por muy diversas razones, entre las que se encuentran las siguientes:
  - Para simplificar el proceso de realización de llamadas a funciones externas, convirtiendo las llamadas a funciones nativas de la aplicación que llama en un formato comprensible por la aplicación externa.
  - Para evitar limitaciones de seguridad u otras restricciones que impiden a la aplicación que llama comunicarse directamente con la aplicación externa.
- Serializar: convertir objetos o valores de datos a un formato que se pueda utilizar para transmitir los valores en mensajes entre dos sistemas de programación, como a través de Internet o entre dos aplicaciones diferentes que se ejecutan en un mismo equipo.

Ejecución de los ejemplos del capítulo

A medida que progrese en el estudio de este capítulo, podría desear probar los listados de código de ejemplo. Muchos de los listados de código del capítulo son pequeños fragmentos de código incluidos con fines de demostración, no ejemplos completos o código de comprobación de valores. Como el uso de la API externa requiere (por definición) escribir código ActionScript así como código en una aplicación contenedora, para probar los ejemplos hay que crear un contenedor (por ejemplo, una página Web que contenga el archivo SWF) y utilizar los listados de código para interactuar con el contenedor.

Para probar un ejemplo de comunicación entre ActionScript y JavaScript:

1. Cree un nuevo documento utilizando la herramienta de edición de Flash y guárdelo en su equipo.
2. En el menú principal, elija Archivo > Configuración de publicación.
3. En el cuadro de diálogo Configuración de publicación, en la ficha Formatos, compruebe que sólo están activadas las casillas de verificación HTML y Flash.
4. Haga clic en el botón Publicar. Esto genera un archivo SWF y un archivo HTML en la misma carpeta y con el mismo nombre que utilizó para guardar el documento de Haga clic en Aceptar para cerrar el cuadro de diálogo Configuración de publicación.
5. Desactive la casilla de verificación HTML. Una vez generada la página HTML, va a modificarla para añadir el código JavaScript apropiado. Si desactiva la casilla de verificación HTML, cuando modifique la página HTML, Flash no sobrescribirá los cambios con una nueva página HTML al publicar el archivo SWF.
6. Haga clic en Aceptar para cerrar el cuadro de diálogo Configuración de publicación.

7 Con un editor de HTML o texto, abra el archivo HTML creado por Flash al publicar el archivo SWF. En el código fuente HTML, agregue etiquetas de apertura y cierre `script` y cópielas en el código JavaScript del listado de código de ejemplo:

```
<script>
// add the sample JavaScript code here
</script>
```

8 Guarde el archivo HTML y vuelva a Flash.

9 Seleccione el fotograma clave del Fotograma 1 de la línea de tiempo y abra el panel Acciones.

10 Copie el listado de código ActionScript en el panel Script.

11 En el menú principal, elija Archivo > Publicar para actualizar el archivo SWF con los cambios realizados.

12 Abra la página HTML editada en un navegador Web para verla y probar la comunicación entre ActionScript y la página HTML.

Para probar un ejemplo de comunicación entre ActionScript y un contenedor ActiveX:

1 Cree un nuevo documento utilizando la herramienta de edición de Flash y guárdelo en su equipo. Puede guardarlo en cualquier carpeta en la que la aplicación contenedora espere encontrar el archivo SWF.

2 En el menú principal, elija Archivo > Configuración de publicación.

3 En el cuadro de diálogo Configuración de publicación, en la ficha Formatos, compruebe que sólo está activada la casilla de verificación Flash.

4 En el campo Archivo situado junto a la casilla de verificación Flash, haga clic en el icono de carpeta para seleccionar la carpeta en la que desea que se publique el archivo SWF. Al establecer la ubicación del archivo SWF puede, por ejemplo, mantener el documento de edición en una carpeta y colocar el archivo SWF publicado en otra carpeta, como la carpeta que contiene el código fuente de la aplicación contenedora.

5 Seleccione el fotograma clave del Fotograma 1 de la línea de tiempo y abra el panel Acciones.

6 Copie el listado de código ActionScript del ejemplo en el panel Script.

7 En el menú principal, elija Archivo > Publicar para volver a publicar el archivo SWF.

8 Cree y ejecute la aplicación contenedora para probar la comunicación entre ActionScript y dicha aplicación.

Los dos ejemplos del final de este capítulo son ejemplos completos del uso de la API externa para comunicarse con una página HTML y una aplicación de escritorio escrita en C#, respectivamente. Estos ejemplos incluyen el código completo, incluido el código de comprobación de errores de ActionScript y de la aplicación contenedora que se debe utilizar al escribir código con la API externa. Otro ejemplo completo de uso de la API externa es el ejemplo de la clase ExternalInterface de la Referencia del lenguaje y componentes ActionScript 3.0.

Requisitos y ventajas de la API externa

La API externa es la parte de ActionScript que proporciona un mecanismo de comunicación entre ActionScript y el código que se ejecuta en una "aplicación externa", que actúa a modo de contenedor de Flash Player (normalmente un navegador Web o una aplicación de proyector independiente). En ActionScript 3.0, la funcionalidad de la API externa viene dada por la clase ExternalInterface. En las versiones de Flash Player anteriores a Flash Player 8, se usaba la acción `fscommand()` para llevar a cabo la comunicación con la aplicación contenedora. La clase ExternalInterface sustituye a `fscommand()` y se recomienda su uso para todas las comunicaciones entre JavaScript y ActionScript.

***Nota:** la antigua función `fscommand()` aún está disponible como función a nivel de paquete en el paquete flash.system para los casos en los que resulte necesario usarla (por ejemplo para mantener la compatibilidad con aplicaciones anteriores o para interactuar con una aplicación contenedora de SWF de un tercero o con la versión autónoma de Flash Player).*

La clase ExternalInterface es un subsistema que permite comunicar fácilmente ActionScript y Flash Player con JavaScript en una página HTML o con cualquier aplicación de escritorio que incluya una instancia de Flash Player.

La clase ExternalInterface sólo está disponible en los siguientes casos:

- En todas las versiones compatibles de Internet Explorer para Windows (5.0 y versiones posteriores)
- En una aplicación contenedora, como una aplicación de escritorio que utilice una instancia del control ActiveX de Flash Player
- En cualquier navegador que admita la interfaz NPRuntime, que actualmente incluye Firefox 1.0 y posterior, Mozilla 1.7.5 y posterior, Netscape 8.0 y posterior y Safari 1.3 y posterior.

En todas las demás situaciones (como al ejecutarse en un reproductor autónomo), la propiedad `ExternalInterface.available` devuelve el valor `false`.

Desde ActionScript se puede llamar a una función de JavaScript en la página HTML. La API externa ofrece las siguientes mejoras con respecto a `fscommand()`:

- Se puede utilizar cualquier función de JavaScript, no sólo que se usan con la función `fscommand()`.
- Se puede pasar un número arbitrario de argumentos y los argumentos pueden tener el nombre que se desee; no existe la limitación de pasar un comando con un único argumento de tipo cadena. Esto hace que la API externa sea mucho más flexible que `fscommand()`.
- Se pueden pasar diversos tipos de datos (como Boolean, Number y String); ya no se está limitado a los parámetros de tipo String.
- Se puede recibir el valor de una llamada y ese valor vuelve inmediatamente a ActionScript (como valor devuelto de la llamada que se realiza).

***Importante:** si el nombre asignado a la instancia de Flash Player en una página HTML (el atributo `id` de la etiqueta `object`) incluye un guión (`-`) u otros caracteres definidos como operadores en JavaScript (como `+`, `*`, `/`, `\`, `.`, etc.), las llamadas a ExternalInterface desde ActionScript no funcionarán cuando la página Web contenedora se visualice en Internet Explorer. Asimismo, si las etiquetas HTML que definen la instancia de Flash Player (etiquetas `object` y `embed`) están anidadas en una etiqueta HTML `form`, las llamadas a ExternalInterface desde ActionScript no funcionarán.*

Utilización de la clase ExternalInterface

La comunicación entre ActionScript y la aplicación contenedora puede adoptar una de las siguientes formas: ActionScript puede llamar al código (por ejemplo, una función de JavaScript) definido en el contenedor, o bien, el código del contenedor puede llamar una función de ActionScript que se haya designado como función que es posible llamar. En cualquiera de los casos, se puede enviar la información al código que se está llamando y se pueden devolver los resultados al código que realiza la llamada.

Para facilitar esta comunicación, la clase ExternalInterface incluye dos propiedades estáticas y dos métodos estáticos. Estas propiedades y métodos se utilizan para obtener información acerca de la conexión de la interfaz externa, para ejecutar código en el contenedor desde ActionScript y para hacer que las funciones ActionScript estén disponibles para ser llamadas desde el contenedor.

Obtención de información sobre el contenedor externo

La propiedad `ExternalInterface.available` indica si Flash Player se encuentra en esos momentos en un contenedor que ofrece una interfaz externa. Si la interfaz externa está disponible, esta propiedad es `true`; en caso contrario, es `false`. Antes de utilizar cualquier otra funcionalidad de la clase ExternalInterface, es recomendable comprobar que el contenedor que se está utilizando en esos momentos permite la comunicación a través de la interfaz externa del modo siguiente:

```
if (ExternalInterface.available)
{
    // Perform ExternalInterface method calls here.
}
```

Nota: *la propiedad `ExternalInterface.available` indica si el contenedor que se está utilizando en esos momentos es de un tipo compatible con la conectividad mediante ExternalInterface. No indicará si JavaScript está habilitado en el navegador.*

La propiedad `ExternalInterface.objectID` permite determinar el identificador exclusivo de la instancia de Flash Player (específicamente, el atributo `id` de la etiqueta `object` en Internet Explorer o el atributo `name` de la etiqueta `embed` en los navegadores que usan la interfaz NPRuntime). Este identificador exclusivo representa al documento SWF actual en el navegador y se puede utilizar para hacer referencia al documento SWF (por ejemplo, al llamar a una función de JavaScript en una página HTML contenedora). Cuando el contenedor de Flash Player no es un navegador Web, esta propiedad es `null`.

Llamada a código externo desde ActionScript

El método `ExternalInterface.call()` ejecuta código en la aplicación contenedora. Como mínimo requiere un parámetro, una cadena que contenga el nombre de la función que se va a llamar en la aplicación contenedora. Todos los parámetros adiciones que se pasen al método `ExternalInterface.call()` se pasarán a su vez al contenedor como parámetros de la llamada a la función.

```
// calls the external function "addNumbers"
// passing two parameters, and assigning that function's result
// to the variable "result"
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

Si el contenedor es una página HTML, este método invocará a la función de JavaScript con el nombre especificado, que debe estar definida en un elemento `script` en la página HTML contenedora. El valor devuelto por la función de JavaScript se devuelve a ActionScript.

```
<script language="JavaScript">
    // adds two numbers, and sends the result back to ActionScript
    function addNumbers(num1, num2)
    {
        return (num1 + num2);
    }
</script>
```

Si el contenedor es algún otro contenedor ActiveX, este método hace que el control ActiveX de Flash Player distribuya su evento `FlashCall`. Flash Player serializa en una cadena XML el nombre de la función especificada y todos los parámetros. El contenedor puede acceder a esa información de la propiedad `request` del objeto de evento y utilizarla para determinar el modo en que se ejecutará su propio código. Para devolver un valor a ActionScript, el código del contenedor llama al método `SetReturnValue()` del objeto ActiveX y pasa el resultado (serializado en una cadena XML) como parámetro de ese método. Para obtener más información sobre el formato XML utilizado para esta comunicación, consulte "Formato XML de la API externa" en la página 701.

Tanto si el contenedor es un navegador Web como si otro contenedor ActiveX, en el caso de que se produzca un error en la llamada o el método del contenedor no especifique un valor devuelto, se devolverá `null`. El método `ExternalInterface.call()` emite una excepción SecurityError si el entorno contenedor pertenece a un entorno limitado de seguridad al que no tiene acceso el código que realiza la llamada. Se puede solucionar esta limitación asignando un valor adecuado a `allowScriptAccess` en el entorno contenedor. Por ejemplo, para cambiar el valor de `allowScriptAccess` en una página HTML, es necesario editar el atributo adecuado en las etiquetas `object` y `embed`.

Llamadas a código ActionScript desde el contenedor

Un contenedor sólo puede llamar a código ActionScript que se encuentre en una función; no puede llamar a ningún otro código ActionScript. Para llamar una función de ActionScript desde la aplicación contenedora, debe realizar dos operaciones: registrar la función con la clase ExternalInterface y posteriormente llamarla desde el código del contenedor.

En primer lugar se debe registrar la función de ActionScript para indicar que debe ponerse a disposición del contenedor. Para ello se usa el método `ExternalInterface.addCallback()` del modo siguiente:

```
function callMe(name:String):String
{
    return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

El método `addCallback()` utiliza dos parámetros: El primero, un nombre de función con formato String, es el nombre por el que se conocerá a la función en el contenedor. El segundo parámetro es la función real de ActionScript que se ejecutará cuando el contenedor llame al nombre de función definido. Dado que estos dos nombres son distintos, se puede especificar un nombre de función que se utilizará en el contenedor incluso si la función real de ActionScript tiene un nombre distinto. Esto resulta especialmente útil si no se conoce el nombre de la función (por ejemplo, cuando se especifica una función anónima o cuando la función que ha de llamarse se determina en tiempo de ejecución).

Una vez que la función de ActionScript se registra en la clase ExternalInterface, el contenedor puede llamar a la función. La forma de realizar esto varía según el tipo de contenedor. Por ejemplo, en el código JavaScript en un navegador Web, la función de ActionScript se llama utilizando el nombre de función registrado como si fuese un método del objeto del navegador de Flash Player (es decir, un método del objeto JavaScript que representa a la etiqueta `object` o `embed)`. Dicho de otro modo, se pasan los parámetros y se devuelve el resultado como si se llamase a una función local.

```
<script language="JavaScript">
    // callResult gets the value "busy signal"
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

Por otra parte, al llamar a una función de ActionScript en un archivo SWF que se ejecuta en una aplicación de escritorio, el nombre de función registrado y los parámetros deben serializarse en una cadena con formato XML. A continuación, se efectúa la llamada llamando al método `CallFunction()` del control ActiveX con la cadena XML como parámetro. Para obtener más información sobre el formato XML utilizado para esta comunicación, consulte "Formato XML de la API externa" en la página 701.

En cualquiera de los dos casos, el valor devuelto por la función de ActionScript se pasa al código del contenedor, ya sea directamente como un valor si la llamada se origina en el código JavaScript de un navegador o serializado como una cadena con formato XML si la llamada se origina en un contenedor ActiveX.

Formato XML de la API externa

En la comunicación entre ActionScript y la aplicación que aloja el control ActiveX de Shockwave Flash se utiliza un formato XML específico para codificar las llamadas a las funciones y los valores. El formato XML usado por la API externa tiene dos partes. Se utiliza un formato para representar las llamadas a funciones. El otro se utiliza para representar valores individuales; este formato se emplea para parámetros de funciones y para los valores devueltos por las funciones. El formato XML para llamadas a funciones se utiliza para llamadas desde y hacia ActionScript. En el caso de una llamada a una función desde ActionScript, Flash Player pasa los datos XML al contenedor; cuando se trata de una llamada desde el contenedor, Flash Player espera que la aplicación contenedora pase una cadena XML con este formato. El siguiente fragmento XML muestra un ejemplo de llamada a función con formato XML:

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

El nodo raíz es `invoke`, Cuenta con dos atributos: `name` indica el nombre de la función que se va a llamar y `returntype` siempre es `xml`. Si la llamada a la función incluye parámetros, el nodo `invoke` tendrá un nodo secundario `arguments`, cuyos nodos secundarios serán los valores de los parámetros a los que se ha dado el formato de valor individual que se explica a continuación.

Los valores individuales, incluidos los parámetros de las funciones y los valores devueltos por éstas, utilizan un esquema de formato que, además de los valores reales, incluye información sobre los tipos de datos. En la siguiente tabla se indican las clases de ActionScript y el formato XML utilizado para codificar los valores de ese tipo de datos:

Clase/Valor de ActionScript	Clase/Valor de C#	Formato	Comentarios
null	null	<null/>	
Boolean `true`	bool true	<true/>	
Boolean `false`	bool false	<false/>	
String	cadena	<string>valor de cadena</string>	
Number, int, uint	single, double, int, uint	`<number>27.5</number>` `<number>-12</number>`	

Clase/Valor de ActionScript	Clase/Valor de C#	Formato	Comentarios

Array (los elementos pueden ser de varios tipos)	Un conjunto que admite elementos de varios tipos, como ArrayList u object[]	`<array>`
		`<property id="0">`
		`<number>27.5</number>`
		`</property>`
		`<property id="1">`
		`<string>Hello there!</string>`
		`</property>`
		`...`
		`</array>`
			El nodo `property` define elementos individuales y el atributo `id` es el índice numérico de base cero.

Object	Un diccionario con claves de tipo cadena y valores de objeto, como un objeto HashTable con claves de tipo cadena	`<object>`
		`<property id="name">`
		`<string>John Doe</string>`
		`</property>`
		`<property id="age">`
		`<string>33</string>`
		`</property>`
		`...`
		`</object>`
			El nodo `property` define propiedades individuales y el atributo `id` es el nombre de la propiedad (una cadena).

Otras clases incorporadas o personalizadas		`<null/> or` `<object></object>`	ActionScript codifica otros objetos como null o como un objeto vacío. En ambos casos, los valores de la propiedad se pierden.

Nota: *a modo de ejemplo, esta tabla muestra las clases de C# equivalentes además de las clases de ActionScript; no obstante, la API externa se puede usar para comunicarse con cualquier lenguaje de programación o entorno de tiempo de ejecución compatible con los controles ActiveX y no se limita a las aplicaciones C#.*

Cuando se crean aplicaciones propias utilizando la API externa con una aplicación contenedora ActiveX, suele resultar cómodo escribir un proxy encargado de la tarea de convertir las llamadas a funciones nativas en el formato XML serializado. Para ver un ejemplo de una clase proxy escrita en C#, consulte "Dentro de la clase ExternalInterfaceProxy" en la página 712.

Ejemplo: Utilización de la API externa con una página Web contenedora

En esta aplicación de ejemplo se muestran técnicas apropiadas para establecer una comunicación entre ActionScript y JavaScript en un navegador Web en el contexto de una aplicación de mensajería instantánea que permite a una persona charlar consigo misma (de ahí el nombre de la aplicación: Introvert IM). Los mensajes se envían entre un formulario HTML en la página Web y una interfaz SWF mediante la API externa. Este ejemplo ilustra las siguientes técnicas:

- Iniciación adecuada de la comunicación comprobando que el navegador está listo para ello antes de establecer la comunicación.
- Comprobación de que el contenedor es compatible con la API externa.
- Llamada a funciones de JavaScript desde ActionScript, paso de parámetros y recepción de valores como respuesta.
- Preparación de métodos de ActionScript para que puedan ser llamados y ejecución de dichas llamadas.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Introvert IM se encuentran en la carpeta Samples/IntrovertIM_HTML. La aplicación consta de los siguientes archivos:

Archivo	Descripción
IntrovertIMApp.fla o IntrovertIMApp.mxml	El archivo de aplicación principal para Flash (FLA) o Flex (MXML).
com/example/programmingas3/introvertIM/IMManager.as	La clase que establece y gestiona la comunicación entre ActionScript y el contenedor.
com/example/programmingas3/introvertIM/IMMessageEvent.as	Tipo de evento personalizado distribuido por la clase IMManager cuando se recibe un mensaje del contenedor.
com/example/programmingas3/introvertIM/IMStatus.as	Enumeración cuyos valores representan los distintos estados de "disponibilidad" que se pueden seleccionar en la aplicación.
html-flash/IntrovertIMApp.html o html-template/index.template.html	La página HTML de la aplicación para Flash (html-flash/IntrovertIMApp.html) o la plantilla que se utiliza para crear la página HTML contenedora de la aplicación para Adobe Flex (html-template/index.template.html). Este archivo contiene todas las funciones de JavaScript que constituyen la parte contenedora de la aplicación.

Preparación de la comunicación ActionScript-navegador

Uno de los usos más frecuentes de la API externa es el de permitir a las aplicaciones ActionScript comunicarse con un navegador Web. Gracias a la API externa, los métodos de ActionScript pueden llamar a código escrito en JavaScript y viceversa. A causa de la complejidad de los navegadores y al modo en el que representan internamente las páginas, no hay manera de garantizar que un documento SWF registrará sus funciones de repetición de llamada antes de que se ejecute el primer fragmento de código JavaScript de la página HTML. Por ese motivo, antes de llamar a las funciones del documento SWF desde JavaScript, es aconsejable que el documento SWF llame siempre a la página HTML para notificarle que está listo para aceptar conexiones.

Por ejemplo, mediante una serie de pasos efectuados por la clase IMManager, Introvert IM determina si el navegador está listo para la comunicación y prepara el archivo SWF para la comunicación. El último paso, que consiste en determinar cuándo está listo el navegador para la comunicación, ocurre en el constructor de IMManager del modo siguiente:

```
public function IMManager(initialStatus:IMStatus)
{
    _status = initialStatus;

    // Check if the container is able to use the external API.
    if (ExternalInterface.available)
    {
        try
        {
            // This calls the isContainerReady() method, which in turn calls
            // the container to see if Flash Player has loaded and the container
            // is ready to receive calls from the SWF.
            var containerReady:Boolean = isContainerReady();
            if (containerReady)
            {
                // If the container is ready, register the SWF's functions.
                setupCallbacks();
            }
            else
            {
                // If the container is not ready, set up a Timer to call the
                // container at 100ms intervals. Once the container responds that
                // it's ready, the timer will be stopped.
                var readyTimer:Timer = new Timer(100);
                readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                readyTimer.start();
            }
        }
        ...
    }
    else
    {
        trace("External interface is not available for this container.");
    }
}
```

En primer lugar, el código comprueba si la API externa está disponible en el contendor actual utilizando la propiedad `ExternalInterface.available`. En caso afirmativo, inicia el proceso de establecimiento de la comunicación. Dado que pueden producirse excepciones de seguridad y otros errores al intentar comunicarse con una aplicación externa, el código se escribe dentro de un bloque `try` (los bloques `catch` correspondientes se han omitido del listado en aras de la brevedad).

A continuación, el código llama al método `isContainerReady()`, que se muestra aquí:

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

El método `isContainerReady()` usa a su vez el método `ExternalInterface.call()` para llamar a la función de JavaScript `isReady()` del modo siguiente:

```
<script language="JavaScript">
<!--
// ------- Private vars -------
var jsReady = false;
...
// ------- functions called by ActionScript -------
// called to check if the page has initialized and JavaScript is available
function isReady()
{
    return jsReady;
}
...
// called by the onload event of the <body> tag
function pageInit()
{
    // Record that JavaScript is ready to go.
    jsReady = true;
}
...
//-->
</script>
```

La función `isReady()` simplemente devuelve el valor de la variable `jsReady` . Inicialmente, la variable tiene el valor `false`; una vez que el evento `onload` de la página Web se ha activado, el valor de la variable cambia a `true`. Dicho de otro modo, si ActionScript llama a la función `isReady()` antes de que se cargue la página, JavaScript devuelve `false` a `ExternalInterface.call("isReady")` y, por lo tanto, el método `isContainerReady()` de ActionScript devuelve `false`. Una vez que la página se ha cargado, la función `isReady()` de JavaScript devuelve `true`, de modo que el método `isContainerReady()` de ActionScript también devuelve `true`.

De vuelta en el constructor de IMManager ocurre una de las dos cosas siguientes, en función de la disponibilidad del contenedor. Si `isContainerReady()` devuelve `true`, el código simplemente llama al método `setupCallbacks()`, que completa el proceso de configuración de la comunicación con JavaScript. Por otra parte, si `isContainerReady()` devuelve `false`, el proceso se pone en espera. Se crea un objeto Timer y se le indica que llame al método `timerHandler()` cada 100 milisegundos del modo siguiente:

```
private function timerHandler(event:TimerEvent):void
{
    // Check if the container is now ready.
    var isReady:Boolean = isContainerReady();
    if (isReady)
    {
        // If the container has become ready, we don't need to check anymore,
        // so stop the timer.
        Timer(event.target).stop();
        // Set up the ActionScript methods that will be available to be
        // called by the container.
        setupCallbacks();
    }
}
```

Cada vez que se llama al método `timerHandler()`, éste vuelve a comprobar el resultado del método `isContainerReady()`. Una vez que se inicializa el contenedor, el método devuelve `true` . En ese momento, el código detiene el temporizador y llama al método `setupCallbacks()` para terminar el proceso de configuración de las comunicaciones con el navegador.

Exposición de los métodos de ActionScript a JavaScript

Como se muestra en el ejemplo anterior, una vez que el código determina que el navegador está listo, se llama al método `setupCallbacks()`. Este método prepara a ActionScript para recibir llamadas desde JavaScript tal y como se muestra a continuación:

```
private function setupCallbacks():void
{
    // Register the SWF client functions with the container
    ExternalInterface.addCallback("newMessage", newMessage);
    ExternalInterface.addCallback("getStatus", getStatus);
    // Notify the container that the SWF is ready to be called.
    ExternalInterface.call("setSWFIsReady");
}
```

El método `setCallBacks()` finaliza la tarea de preparar la comunicación con el contenedor llamando a `ExternalInterface.addCallback()` para registrar los dos métodos que estarán disponibles para ser llamados desde JavaScript. En este código, el primer parámetro (el nombre por el que el método se conoce en JavaScript, `"newMessage"` y `"getStatus"`) es el mismo que el nombre del método en ActionScript (en este caso no había ninguna ventaja en usar nombres distintos, así que se reutilizó el mismo nombre por simplificar). Por último, el método `ExternalInterface.call()` se utiliza para llamar a la función `setSWFIsReady()` de JavaScript, que notifica al contenedor que las funciones ActionScript se han registrado.

Comunicación desde ActionScript al navegador

La aplicación Introvert IM presenta toda una gama de ejemplos de llamadas a funciones JavaScript en la página contenedora. En el caso más simple (un ejemplo del método `setupCallbacks()`), se llama a la función `setSWFIsReady()` de JavaScript sin pasarle ningún parámetro ni recibir ningún valor devuelto:

```
ExternalInterface.call("setSWFIsReady");
```

En otro ejemplo del método `isContainerReady()`, ActionScript llama a la función `isReady()` y recibe un valor Boolean como respuesta:

```
var result:Boolean = ExternalInterface.call("isReady");
```

También se puede pasar parámetros a funciones de JavaScript a través de la API externa. Esto puede observarse, por ejemplo, en el método `sendMessage()` de la clase IMManager, al que se llama cuando el usuario envía un mensaje nuevo a su "interlocutor":

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

De nuevo se utiliza `ExternalInterface.call()` para llamar a la función de JavaScript designada, que notifica al navegador que hay un nuevo mensaje. Además, el mensaje en sí se pasa como un parámetro adicional a `ExternalInterface.call()` y, por lo tanto, se pasa como un parámetro a la función de JavaScript `newMessage()`.

Llamadas a código de ActionScript desde JavaScript

La comunicación suele ser bidireccional y, en este sentido, la aplicación Introvert IM no es ninguna excepción. No sólo el cliente de mensajería instantánea de Flash Player llama a JavaScript para enviar mensajes, sino que también el formulario HTML llama al código JavaScript para enviar mensajes y recibir información del archivo SWF. Por ejemplo, cuando el archivo SWF notifica al contenedor que ha terminado de establecer contacto y que está listo para empezar a comunicarse, lo primero que hace el navegador es llamar al método `getStatus()` de la clase IMManager para recuperar el estado de disponibilidad inicial del usuario desde el cliente de mensajería instantánea de SWF. Esto se lleva a cabo en la página Web, en la función `updateStatus()`, del modo siguiente:

```
<script language="JavaScript">
...
function updateStatus()
{
    if (swfReady)
    {
        var currentStatus = getSWF("IntrovertIMApp").getStatus();
        document.forms["imForm"].status.value = currentStatus;
    }
}
...
</script>
```

El código comprueba el valor de la variable `swfReady`, que determina si el archivo SWF ha notificado al navegador que éste ha registrado sus métodos en la clase ExternalInterface. Si el archivo SWF está listo para recibir comunicaciones, la siguiente línea (`var currentStatus =` ...) llama al método `getStatus()` de la clase IMManager. En esta línea de código ocurren tres cosas:

- Se llama a la función `getSWF()` de JavaScript, que devuelve una referencia al objeto JavaScript que representa al archivo SWF. El parámetro pasado a `getSWF()` determina qué objeto de navegador se devuelve en caso de que haya más un archivo SWF en una página HTML. El valor pasado a ese parámetro debe coincidir con el atributo `id` de la etiqueta `object` y el atributo `name` de la etiqueta `embed` utilizada para incluir el archivo SWF.
- Se utiliza la referencia al archivo SWF para llamar al método `getStatus()` como si fuese un método del objeto SWF. En este caso, se utiliza el nombre de función "`getStatus`", ya que ese el nombre con el que se ha registrado la función de ActionScript mediante `ExternalInterface.addCallback()`.
- El método `getStatus()` de ActionScript devuelve un valor, que se asigna a la variable `currentStatus`, que se asigna a su vez como contenido (la propiedad `value`) del campo de texto `status`.

Nota: *si sigue el código, probablemente se habrá dado cuenta de que en el código fuente de la función `updateStatus()`, la línea de código que llama a la función `getSWF()` en realidad se escribe del modo siguiente: var currentStatus = getSWF("${application}").getStatus(); El texto `${application}` es un marcador de posición de la plantilla de la página HTML; cuando Adobe Flex Builder 3 genera la página HTML real de la aplicación, este texto se sustituye por el texto utilizado como atributo `id` de la etiqueta `object` y como atributo `name` de la etiqueta `embed` (`IntrovertIMApp` en el ejemplo). Ese es el valor que espera la función `getSWF()`.*

En la función `sendMessage()` de JavaScript se muestra el modo de pasar un parámetro a una función de ActionScript `sendMessage()` es la función llamada cuando el usuario pulsa el botón Send de la página HTML).

```
<script language="JavaScript">
...
function sendMessage(message)
{
    if (swfReady)
    {
        ...
        getSWF("IntrovertIMApp").newMessage(message);
    }
}
...
</script>
```

El método `newMessage()` de ActionScript espera un parámetro, de modo que la variable `message` de JavaScript se pasa a ActionScript usándola como parámetro en la llamada al método `newMessage()` en el código JavaScript.

Detección del tipo de navegador

A causa de las diferencias en el modo en el que los navegadores acceden al contenido, es importante usar siempre JavaScript para detectar qué navegador está utilizando el usuario y para acceder a la película de acuerdo con su sintaxis específica, empleando el objeto window o document, según se muestra en la función `getSWF()` de JavaScript en este ejemplo:

```
<script language="JavaScript">
...
function getSWF(movieName)
{
    if (navigator.appName.indexOf("Microsoft") != -1)
    {
        return window[movieName];
    }
    else
    {
        return document[movieName];
    }
}
...
</script>
```

Si el script no detecta el tipo de navegador del usuario, puede producirse un comportamiento inesperado al reproducir archivos SWF en un contenedor HTML.

Ejemplo: Utilización de la API externa con un contenedor ActiveX

En este ejemplo se muestra el uso de la API externa para establecer una comunicación entre ActionScript y una aplicación de escritorio que usa el control ActiveX. Para el ejemplo se reutiliza la aplicación Introvert IM, incluido el código ActionScript e incluso el mismo archivo SWF y, por lo tanto, no se describe el uso de la API externa en ActionScript. Para entender este ejemplo, resultará de utilidad estar familiarizado con el anterior.

La aplicación de escritorio de este ejemplo se ha escrito en C# con Microsoft Visual Studio .NET. El análisis se centra en las técnicas específicas para trabajar con la API externa usando el control ActiveX. En este ejemplo se muestran los siguientes procedimientos:

- Llamar a funciones ActionScript desde una aplicación de escritorio que aloja el control ActiveX de Flash Player.
- Recibir llamadas a funciones desde ActionScript y procesarlas en un contenedor ActiveX.
- Utilizar una clase proxy para ocultar los detalles del formato XML serializado que utiliza Flash Player para los mensajes enviados a un contenedor ActiveX.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos C# de Introvert IM se encuentran en la carpeta Samples/IntrovertIM_CSharp. La aplicación consta de los siguientes archivos:

Archivo	Descripción
AppForm.cs	El archivo de aplicación principal con la interfaz Windows Forms en C#.
bin/Debug/IntrovertIMApp.swf	El archivo SWF que carga la aplicación.
ExternalInterfaceProxy/ExternalInterfaceProxy.cs	La clase que actúa como envolvente del control ActiveX para la comunicación con la interfaz externa. Proporciona mecanismos para llamar y recibir llamadas desde ActionScript.
ExternalInterfaceProxy/ExternalInterfaceSerializer.cs	La clase que realiza la tarea de convertir los mensajes en formato XML de Flash Player en objetos .NET.
ExternalInterfaceProxy/ExternalInterfaceEventArgs.cs	Este archivo define dos tipos C# (clases): una clase de delegado personalizada y una clase de argumentos de evento, ambas utilizadas por la clase ExternalInterfaceProxy para notificar a un detector la existencia de una llamada a función desde ActionScript.
ExternalInterfaceProxy/ExternalInterfaceCall.cs	Esta clase es un valor de objeto que representa una llamada a una función desde ActionScript al contenedor ActiveX, con propiedades para el nombre de función y los parámetros.
bin/Debug/IntrovertIMApp.swf	El archivo SWF que carga la aplicación.
obj/AxInterop.ShockwaveFlashObjects.dll, obj/Interop.ShockwaveFlashObjects.dll	Ensamblados de envolventes creados por Visual Studio .NET que son necesarios para acceder al control ActiveX de Flash Player (Adobe Shockwave® Flash) desde el código administrado.

Información general de la aplicación Introvert IM escrita en C#

Esta aplicación de ejemplo representa dos programas cliente de mensajería instantánea (uno dentro de un archivo SWF y otro creado con Windows Forms) que se comunican entre sí. La interfaz de usuario incluye una instancia del control ActiveX de Shockwave Flash, dentro del cual se carga el archivo SWF que contiene el cliente de mensajería instantánea de ActionScript. La interfaz también incluye varios campos de texto que conforman el cliente de mensajería instantánea de Windows Forms: un campo para introducir mensajes (`MessageText`), otro que muestra la transcripción de los mensajes enviados entre clientes (`Transcript`) y un tercero (`Status`) que indica el estado de disponibilidad tal y como se establece en el cliente de mensajería instantánea del archivo SWF.

Inclusión del control ActiveX de Shockwave Flash

Para incluir el control ActiveX de Shockwave Flash en una aplicación de Windows Forms, en primer lugar es necesario añadirlo al cuadro de herramientas de Microsoft Visual Studio.

Para añadir el control al cuadro de herramientas:

1 Abra el cuadro de herramientas de Visual Studio.

2 Haga clic con el botón derecho en la sección Windows Forms en Visual Studio 2003 o en cualquier sección en Visual Studio 2005. En el menú contextual, seleccione Agregar o quitar elementos en Visual Studio 2003 (Elegir elementos... en Visual Studio 2005).

Se abrirá el cuadro de diálogo Personalizar cuadro de herramientas (2003)/Elegir elementos del cuadro de herramientas (2005).

3 Seleccione la ficha Componentes COM, en la que aparecen todos los componentes COM disponibles en el equipo, incluido el control ActiveX de Flash Player.

4 Desplácese hasta el objeto Shockwave Flash y selecciónelo.

Si este elemento no aparece en la lista, compruebe que el control ActiveX de Flash Player está instalado en el equipo.

Aspectos básicos de la comunicación entre ActionScript y el contenedor ActiveX

El funcionamiento de la comunicación a través de la API externa con una aplicación contenedora ActiveX es igual que el del la comunicación con un navegador Web, con una diferencia importante. Según se expuso anteriormente, cuando ActionScript se comunica con un navegador Web, por lo que respecta al desarrollador, las funciones se llaman directamente; los detalles sobre el modo en que se da formato a las llamadas a funciones y a las respuestas para pasarlas entre el reproductor y el navegador quedan ocultas. No obstante, cuando se utiliza la API externa para comunicarse con una aplicación contenedora ActiveX, Flash Player envía mensajes (llamadas a funciones y valores devueltos) a la aplicación en un formato XML específico y espera que las llamadas a funciones y valores devueltos por la aplicación contenedora tengan el mismo formato XML. El desarrollador de la aplicación contenedora ActiveX debe escribir código que entienda y pueda crear llamadas a funciones y respuestas con el formato adecuado.

El ejemplo de Introvert IM en C# incluye un conjunto de clases que permiten evitar la tarea de dar formato a los mensajes; en vez de eso, es posible trabajar con tipos de datos estándar al llamar a funciones ActionScript y recibir llamadas a funciones desde ActionScript. La clase ExternalInterfaceProxy, junto con otras clases auxiliares, proporcionan esta funcionalidad y se pueden reutilizar en cualquier proyecto .NET para facilitar la comunicación con la API externa.

Las siguientes secciones de código son extractos del formulario de aplicación principal (AppForm.cs) que muestran la interacción simplificada que se logra utilizando la clase ExternalInterfaceProxy:

```
public class AppForm : System.Windows.Forms.Form
{
    ...
    private ExternalInterfaceProxy proxy;
    ...
    public AppForm()
    {
        ...
        // Register this app to receive notification when the proxy receives
        // a call from ActionScript.
        proxy = new ExternalInterfaceProxy(IntrovertIMApp);
        proxy.ExternalInterfaceCall += new
ExternalInterfaceCallEventHandler(proxy_ExternalInterfaceCall);
        ...
    }
    ...
```

La aplicación declara y crea una instancia de ExternalInterfaceProxy llamada `proxy`, que pasa una referencia al control ActiveX de Shockwave Flash que se encuentra en la interfaz de usuario (`IntrovertIMApp`). A continuación, el código registra el método `proxy_ExternalInterfaceCall()` para recibir el evento `ExternalInterfaceCall` del proxy. La clase ExternalInterfaceProxy distribuye este evento cuando llega una llamada a una función desde Flash Player. Al suscribirse a este evento, el código C# puede recibir llamadas a funciones (y responderlas) desde ActionScript.

Cuando llega una llamada a una función desde ActionScript, la instancia de ExternalInterfaceProxy (`proxy`) recibe la llamada, convierte el formato XML y notifica a los objetos que son detectores del evento `ExternalInterfaceCall` de proxy. En el caso de la clase AppForm, el método `proxy_ExternalInterfaceCall()` gestiona ese evento del modo siguiente:

```
/// <summary>
/// Called by the proxy when an ActionScript ExternalInterface call
/// is made by the SWF
/// </summary>
private object proxy_ExternalInterfaceCall(object sender, ExternalInterfaceCallEventArgs e)
{
    switch (e.FunctionCall.FunctionName)
    {
        case "isReady":
            return isReady();
        case "setSWFIsReady":
            setSWFIsReady();
            return null;
        case "newMessage":
            newMessage((string)e.FunctionCall.Arguments[0]);
            return null;
        case "statusChange":
            statusChange();
            return null;
        default:
            return null;
    }
}
...
```

El método pasa una instancia de ExternalInterfaceCallEventArgs llamada `e` en este ejemplo. Ese objeto, a su vez, tiene una propiedad `FunctionCall` que es una instancia de la clase ExternalInterfaceCall.

Una instancia de ExternalInterfaceCall es un valor de objeto sencillo con dos propiedades. La propiedad `FunctionName` contiene el nombre de la función especificado en la sentencia `ExternalInterface.Call()` de ActionScript. Si se añaden más parámetros en ActionScript, éstos se incluirán en la propiedad `Arguments` del objeto ExternalInterfaceCall. En este caso, el método que gestiona el evento es simplemente una sentencia `switch` que actúa como un controlador de tráfico. El valor de la propiedad `FunctionName` (`e.FunctionCall.FunctionName`) determina el método de la clase AppForm al que se llamará.

Las ramas de la sentencia `switch` del listado de código anterior muestran situaciones comunes de llamadas a métodos. Por ejemplo, todo método debe devolver un valor a ActionScript (por ejemplo, la llamada al método `isReady()`) o bien devolver `null` (según se muestra en las demás llamadas a métodos). En la llamada al método `newMessage()` (que pasa un parámetro `e.FunctionCall.Arguments[0]`, es decir, el primer elemento del conjunto `Arguments`) se muestra el acceso a los parámetros pasados desde ActionScript.

Llamar a una función de ActionScript desde C# utilizando la clase ExternalInterfaceProxy es aún más sencillo que recibir una llamada a una función desde ActionScript. Para llamar a una función de ActionScript se utiliza el método `Call()` de la instancia de ExternalInterfaceProxy de la siguiente forma:

```
    /// <summary>
    /// Called when the "Send" button is pressed; the value in the
    /// MessageText text field is passed in as a parameter.
    /// </summary>
    /// <param name="message">The message to send.</param>
    private void sendMessage(string message)
    {
        if (swfReady)
        {
            ...
            // Call the newMessage function in ActionScript.
            proxy.Call("newMessage", message);
        }
    }
    ...
    /// <summary>
    /// Call the ActionScript function to get the current "availability"
    /// status and write it into the text field.
    /// </summary>
    private void updateStatus()
    {
        Status.Text = (string)proxy.Call("getStatus");
    }
    ...
}
```

Como muestra este ejemplo, el método `Call()` de la clase ExternalInterfaceProxy es muy similar a su homólogo de ActionScript, `ExternalInterface.Call()`. El primer parámetro es una cadena, el nombre de la función que se va a llamar. Los parámetros adiciones (que no se muestran aquí) se pasan también a la función de ActionScript. Si la función de ActionScript devuelve un valor, éste se devuelve mediante el método `Call()` (como puede verse en el ejemplo anterior).

Dentro de la clase ExternalInterfaceProxy

Puede que no siempre sea práctico utilizar un envolvente de proxy en torno al control ActiveX o bien el desarrollador puede desear escribir su propia clase proxy (por ejemplo, en un lenguaje de programación diferente u orientado a una plataforma distinta). Si bien aquí no se explican todos los detalles sobre la creación de una clase proxy, resulta útil comprender el funcionamiento interno de la clase proxy en este ejemplo.

Se usa el método `CallFunction()` del control ActiveX de Shockwave Flash para llamar a una función de ActionScript desde el contenedor ActiveX utilizando la API externa. Esto se muestra en el siguiente extracto del método `Call()` de la clase ExternalInterfaceProxy:

```
// Call an ActionScript function on the SWF in "_flashControl",
// which is a Shockwave Flash ActiveX control.
string response = _flashControl.CallFunction(request);
```

En este fragmento de código, `_flashControl` es el control ActiveX de Shockwave Flash. Para realzar las llamadas a funciones ActionScript se utiliza el método `CallFunction()`. Este método acepta un parámetro (`request` en el ejemplo), que es una cadena que contiene instrucciones en formato XML entre las que se incluye el nombre de la función de ActionScript que se llamará y los correspondientes parámetros. Todos los valores devueltos desde ActionScript se codifican como una cadena con formato XML y se envían como el valor devuelto de la llamada a `CallFunction()`. En este ejemplo, la cadena XML se almacena en la variable `response`.

Recibir una llamada a una función desde ActionScript es un proceso compuesto de varios pasos. Las llamadas a funciones desde ActionScript hacen que el control ActiveX de Shockwave Flash distribuya su evento FlashCall, de modo que una clase (como la clase ExternalInterfaceProxy) diseñada para recibir llamadas desde un archivo SWF tiene que definir un controlador para ese evento. En la clase ExternalInterfaceProxy, la función de controlador de eventos se llama `_flashControl_FlashCall()` y está registrada para detectar el evento en el constructor de la clase del siguiente modo:

```
private AxShockwaveFlash _flashControl;

public ExternalInterfaceProxy(AxShockwaveFlash flashControl)
{
    _flashControl = flashControl;
    _flashControl.FlashCall += new
_IShockwaveFlashEvents_FlashCallEventHandler(_flashControl_FlashCall);
}
...
private void _flashControl_FlashCall(object sender, _IShockwaveFlashEvents_FlashCallEvent e)
{
    // Use the event object's request property ("e.request")
    // to execute some action.
    ...
    // Return a value to ActionScript;
    // the returned value must first be encoded as an XML-formatted string.
    _flashControl.SetReturnValue(encodedResponse);
}
```

El objeto de evento (`e`) tiene una propiedad `request` (`e.request`) que es una cadena que contiene información en formato XML acerca de la llamada a la función, como el nombre de la función y los parámetros. El contenedor puede utilizar esta información para determinar qué código se debe ejecutar. En la clase ExternalInterfaceProxy, request se convierte del formato XML a un objeto ExternalInterfaceCall, que proporciona la misma información con un formato más accesible. El método `SetReturnValue()` del control ActiveX se utiliza para devolver el resultado de una función al elemento que origina la llamada en ActionScript; de nuevo, el parámetro del resultado debe estar codificado en el formato XML utilizado por la API externa.

En la comunicación entre ActionScript y la aplicación que aloja el control ActiveX de Shockwave Flash se utiliza un formato XML específico para codificar las llamadas a las funciones y los valores. En el ejemplo en C# de Introvert IM, la clase ExternalInterfaceProxy posibilita que el código del formulario de la aplicación opere directamente sobre los valores enviados o recibidos desde ActionScript e ignore los detalles del formato XML utilizado por Flash Player. Para lograrlo, la clase ExternalInterfaceProxy usa los métodos de la clase ExternalInterfaceSerializer para convertir los mensajes XML en objetos .NET. La clase ExternalInterfaceSerializer tiene cuatro métodos públicos:

- `EncodeInvoke()`: codifica el nombre de una función y una lista de argumentos ArrayList de C# con el formato XML adecuado.
- `EncodeResult()`: codifica un valor de resultado con el formato XML apropiado.
- `DecodeInvoke()`: descodifica una llamada de función desde ActionScript. La propiedad `request` del objeto de evento FlashCall se pasa al método `DecodeInvoke()`, que convierte la llamada en un objeto ExternalInterfaceCall.
- `DecodeResult()`: descodifica los datos XML recibidos como resultado de llamar a una función de ActionScript.

Estos métodos codifican valores de C# en el formato XML de la API externa y descodifica el XML para transformarlo en objetos de C#. Para obtener información sobre el formato XML utilizado por Flash Player, consulte “Formato XML de la API externa” en la página 701.

Capítulo 32: Seguridad de Flash Player

La seguridad es una de las principales preocupaciones de Adobe, los usuarios, los propietarios de sitios Web y los desarrolladores de contenido. Por este motivo, Adobe® Flash® Player incluye un conjunto de reglas de seguridad y controles para proteger al usuario, al propietario del sitio Web y al desarrollador de contenido. En este capítulo se describe cómo trabajar con el modelo de seguridad de Player durante el desarrollo de aplicaciones Flash. A no ser que se indique lo contrario, se da por hecho que todos los archivos SWF mencionados en este capítulo se publican con ActionScript 3.0 y se ejecutan en Flash Player 9.0.124.0 o posterior.

El objetivo de este capítulo es proporcionar información general sobre la seguridad, por lo que no se intentará explicar de forma exhaustiva todos los detalles de implementación, escenarios de uso o ramificaciones para utilizar determinadas API. Para ver una descripción detallada de los conceptos de seguridad de Flash Player, consulte el tema de seguridad del Flash Player Developer Center (centro de desarrolladores de Flash Player) en la dirección www.adobe.com/go/devnet_security_es.

Para obtener más información sobre los problemas de seguridad de Adobe® AIR™, consulte el capítulo sobre seguridad de AIR en www.adobe.com/go/learn_air_flash_es.

Información general sobre la seguridad de Flash Player

La mayor parte de la seguridad de Flash Player se basa en el dominio de origen de los archivos SWF, medios y otros activos cargados. Un archivo SWF de un dominio concreto de Internet, tal como www.ejemplo.com, podrá tener siempre acceso a todos los datos de dicho dominio. Estos activos se incluyen en el mismo grupo de seguridad, conocido como *entorno limitado de seguridad*. (Para obtener más información, consulte “Entornos limitados de seguridad” en la página 716.)

Por ejemplo, un archivo SWF puede cargar archivos SWF, mapas de bits, audio, archivos de texto y otros activos de su propio dominio. Asimismo, la reutilización de scripts siempre está permitida entre dos archivos SWF del mismo dominio, siempre que ambos archivos se hayan escrito utilizando ActionScript 3.0. La *reutilización de scripts* es la capacidad de un archivo SWF para utilizar ActionScript para acceder a las propiedades, los métodos y los objetos de otro archivo SWF.

La reutilización de scripts no es posible entre los archivos SWF escritos en ActionScript 3.0 y los escritos en versiones anteriores de ActionScript; sin embargo, estos archivos pueden comunicarse a través de la clase LocalConnection. Por otro lado, está prohibido de forma predeterminada que un archivo SWF reutilice los scripts de los archivos SWF de ActionScript 3.0 pertenecientes a otros dominios, así como que cargue datos de otros dominios. Sin embargo, este permiso se puede conceder si se llama al método `Security.allowDomain()` del archivo SWF cargado. Para obtener más información, consulte “Reutilización de scripts” en la página 731.

Las reglas de seguridad básicas que siempre se aplican de forma predeterminada son:

- Los recursos del mismo entorno limitado de seguridad tienen acceso libre entre ellos.
- Los archivos SWF de un entorno limitado remoto no tienen nunca acceso a archivos y datos locales.

Flash Player percibe los siguientes dominios como individuales y configura entornos limitados de seguridad individuales para cada uno de ellos:

- `http://example.com`
- `http://www.example.com`

- `http://store.example.com`
- `https://www.example.com`
- `http://192.0.34.166`

Incluso si un dominio con nombre, como http://example.com, corresponde a una dirección IP específica, como http://192.0.34.166, Flash Player configura entornos limitados de seguridad independientes para cada uno de ellos.

Hay dos métodos básicos que puede utilizar un desarrollador para conceder a un archivo SWF el acceso a los activos de entornos limitados ajenos al del archivo SWF:

- El método `Security.allowDomain()` (consulte "Controles de autor (desarrollador)" en la página 725)
- El archivo de política URL (consulte "Controles de sitio Web (archivos de política)" en la página 721)

En el modelo de seguridad de Flash Player, hay una diferencia entre cargar contenido y extraer o acceder a datos. El *contenido* se define como medios, incluidos los medios visuales que Flash Player puede mostrar, audio, video o un archivo SWF que incluye medios mostrados. Se define como *datos* aquello que sólo es accesible para el código ActionScript. El contenido y los datos se cargan de modos diferentes.

- Carga de contenido: el contenido se puede cargar a través de clases como Loader, Sound y NetStream.
- Extracción de datos: los datos se pueden extraer de contenido de medios cargados a través de objetos Bitmap, el método `BitmapData.draw()`, la propiedad `Sound.id3` o el método `SoundMixer.computeSpectrum()`.
- Acceso a datos: se puede acceder a datos directamente cargándolos desde un archivo externo (como un archivo XML) a través de clases como URLStream, URLLoader, Socket y XMLSocket.

El modelo de seguridad de Flash Player define reglas distintas para cargar contenido y acceder a los datos. En general, hay menos restricciones para cargar contenido que para acceder a los datos.

En general, el contenido (archivos SWF, mapas de bits, archivos MP3 y vídeos) puede cargarse desde cualquier origen, pero si procede de un dominio ajeno al del archivo SWF que realiza la carga, se ubicará en un entorno limitado de seguridad independiente.

Hay algunos obstáculos para cargar contenido:

- De forma predeterminada, los archivos SWF locales (cargados desde una dirección fuera de la red como, por ejemplo, el disco duro de un usuario) se clasifican en el entorno limitado local con sistema de archivos. Estos archivos no pueden cargar contenido de la red. Para obtener más información, consulte "Entornos limitados locales" en la página 716.
- Los servidores RTMP (Real-Time Messaging Protocol) pueden limitar el acceso al contenido. Para obtener más información, consulte "Contenido proporcionado a través de servidores RTMP" en la página 731.

Si el medio cargado es una imagen, audio o vídeo, sus datos (ya sean píxeles o sonidos) serán accesibles para un archivo SWF que no pertenezca a su entorno limitado de seguridad sólo si el dominio de dicho archivo SWF se haya incluido en un archivo de política en el dominio de origen del medio. Para obtener información más detallada, consulte "Acceso a medios cargados como datos" en la página 734.

Otros tipos de datos cargados contienen texto o archivos XML, que se cargan con un objeto URLLoader. De nuevo en este caso, para acceder a los datos de otro entorno limitado de seguridad, deben concederse permisos mediante un archivo de política URL en el dominio de origen. Para obtener información más detallada, consulte "Utilización de URLLoader y URLStream" en la página 737.

Entornos limitados de seguridad

Los equipos cliente pueden obtener archivos SWF individuales desde varios orígenes como, por ejemplo, sitios Web externos o un sistema de archivos local. Flash Player asigna individualmente archivos SWF y otros recursos, como objetos compartidos, mapas de bits, sonidos, vídeos y archivos de datos, a entornos limitados de seguridad, en función de su origen, cuando se cargan en Flash Player. En las siguientes secciones se describen las reglas que aplica Flash Player y que controlan dónde puede acceder un archivo SWF de un determinado entorno limitado.

Para obtener más información sobre la seguridad de Flash Player, consulte el tema de seguridad del Flash Player Developer Center (centro de desarrolladores de Flash Player) en www.adobe.com/go/devnet_security_es.

Entornos limitados remotos

Flash Player clasifica los activos (incluidos los archivos SWF) de Internet en entornos limitados independientes, correspondientes a los dominios de origen de sus sitios Web. De forma predeterminada, estos archivos tienen autorización para acceder a cualquier recurso de su propio servidor. Se puede permitir que los archivos SWF accedan a datos adicionales desde otros dominios. Para ello, es necesario conceder permisos de autor y sitio Web explícitos, como los archivos de política URL y el método `Security.allowDomain()`. Para obtener información más detallada, consulte "Controles de sitio Web (archivos de política)" en la página 721 y "Controles de autor (desarrollador)" en la página 725.

Los archivos SWF remotos no pueden cargar archivos ni recursos locales.

Para obtener más información sobre la seguridad de Flash Player, consulte el tema de seguridad del Flash Player Developer Center (centro de desarrolladores de Flash Player) en www.adobe.com/go/devnet_security_es.

Entornos limitados locales

Un *archivo local* es un archivo al que se hace referencia a través del protocolo `file:` o una ruta UNC (convención de nomenclatura universal). Los archivos SWF locales se ubican en uno de los cuatro entornos limitados locales siguientes:

- El entorno limitado local con sistema de archivos: por motivos de seguridad, Flash Player coloca todos los archivos SWF y activos en el entorno limitado local con sistema de archivos, de forma predeterminada. Desde este entorno limitado, los archivos SWF pueden leer archivos locales (por ejemplo, mediante la clase URLLoader), pero no pueden comunicarse con la red de ningún modo. De este modo, el usuario tiene la seguridad de que no se filtran los datos locales a la red ni se comparten de forma indebida.
- El entorno limitado local con acceso a la red: al compilar un archivo SWF, se puede especificar que tenga acceso a la red cuando se ejecute como un archivo local (consulte "Configuración del tipo de entorno limitado de los archivos SWF locales" en la página 717). Estos archivos se colocan en el entorno limitado local con acceso a la red. Los archivos SWF que se asignan al entorno limitado local con acceso a la red pierden el acceso al sistema de archivos local. A cambio, se les permite acceder a los datos de la red. Sin embargo, sigue sin permitirse que un archivo SWF local con acceso a la red lea los datos derivados de la red a menos que se disponga de permisos para ello, a través de un archivo de política URL o una llamada al método `Security.allowDomain()`. Para conceder dicho permiso, un archivo de política URL debe conceder permiso a *todos* los dominios a través de `<allow-access-from domain="*"/>` o `Security.allowDomain("*")`. Para obtener más información, consulte "Controles de sitio Web (archivos de política)" en la página 721 y "Controles de autor (desarrollador)" en la página 725.

- El entorno limitado local de confianza: los archivos SWF locales que los usuarios o programas instaladores registran como archivos de confianza se colocan en el entorno limitado local de confianza. Los administradores del sistema y los usuarios también pueden reasignar un archivo SWF local al entorno limitado local de confianza, o quitarlo de él, en función de consideraciones de seguridad (consulte "Controles de administrador" en la página 718 y "Controles de usuario" en la página 720). Los archivos SWF asignados al entorno limitado local de confianza pueden interactuar con el resto de archivos SWF y cargar datos desde cualquier lugar (remoto o local).
- El entorno limitado de la aplicación AIR: este entorno limitado presenta contenido que se instaló con la aplicación AIR en ejecución. De forma predeterminada, los archivos del entorno limitado de la aplicación AIR pueden reutilizar los scripts de los archivos de cualquier dominio. No obstante, a los archivos que se encuentran fuera del entorno limitado de la aplicación AIR no se les permite reutilizar los scripts del archivo AIR. De forma predeterminada, los archivos del entorno limitado de la aplicación AIR pueden cargar contenido y datos desde cualquier dominio.

Se prohíbe estrictamente la comunicación entre entornos limitados locales con acceso a la red y entornos limitados locales con sistema de archivos, y entre entornos limitados locales con sistema de archivos y entornos limitados remotos. Una aplicación que se ejecute en Flash Player o un usuario o administrador no pueden conceder permiso para permitir dicha comunicación.

La creación de scripts en cualquier sentido entre archivos HTML locales y archivos SWF locales (por ejemplo, mediante la clase ExternalInterface) requiere que los archivos HTML y SWF implicados estén en el entorno limitado local de confianza. El motivo de ello es que los modelos de seguridad local de los navegadores difieren del modelo de seguridad local de Flash Player.

Los archivos SWF incluidos en el entorno limitado local con acceso a la red no pueden cargar archivos SWF en el entorno limitado local con sistema de archivos. Los archivos SWF incluidos en el entorno limitado local con sistema de archivos no pueden cargar archivos SWF en el entorno limitado local con acceso a la red.

Configuración del tipo de entorno limitado de los archivos SWF locales

Se puede configurar un archivo SWF para el entorno limitado local con sistema de archivos o para el entorno limitado local con acceso a la red estableciendo la configuración de publicación del documento en la herramienta de edición

Un usuario final o el administrador de un equipo puede especificar que un archivo SWF local es de confianza y permitir que cargue datos de todos los dominios, tanto locales como de red. Esto se especifica en los directorios Global Flash Player Trust y User Flash Player Trust. Para obtener más información, consulte "Controles de administrador" en la página 718 y "Controles de usuario" en la página 720.

Para obtener más información sobre los entornos limitados locales, consulte "Entornos limitados locales" en la página 716.

La propiedad Security.sandboxType

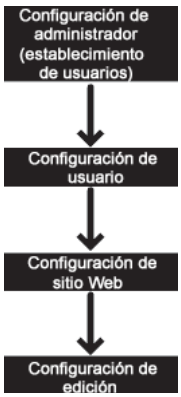
El autor de un archivo SWF puede usar la propiedad `Security.sandboxType` estática de sólo lectura para determinar el tipo de entorno limitado al que Flash Player ha asignado el archivo SWF. La clase Security incluye constantes que representan los valores posibles de la propiedad `Security.sandboxType`, del siguiente modo:

- `Security.REMOTE`: este archivo SWF procede de un URL de Internet y se rige por reglas de entorno limitado basadas en dominios.
- `Security.LOCAL_WITH_FILE`: el SWF es un archivo local y no es de confianza para el usuario. No se ha publicado con una designación de acceso a la red. El archivo SWF puede leer de orígenes de datos locales pero no puede comunicarse con Internet.

- `Security.LOCAL_WITH_NETWORK`: el SWF es un archivo local y no es de confianza para el usuario, pero se ha publicado con una designación de acceso a la red. El archivo SWF se puede comunicar con Internet pero no puede leer en fuentes de datos locales.
- `Security.LOCAL_TRUSTED`: el SWF es un archivo local y el usuario ha determinado que es de confianza, mediante el Administrador de configuración o un archivo de configuración de confianza de Flash Player. El archivo SWF puede leer de orígenes de datos locales y puede comunicarse con Internet.
- `Security.APPLICATION`: el archivo SWF se ejecuta en una aplicación AIR, y se instaló con el paquete (archivo AIR) de la aplicación. De forma predeterminada, los archivos del entorno limitado de la aplicación AIR pueden reutilizar los scripts de los archivos de cualquier dominio. No obstante, a los archivos que se encuentran fuera del entorno limitado de la aplicación AIR no se les permite reutilizar los scripts del archivo AIR. De forma predeterminada, los archivos del entorno limitado de la aplicación AIR pueden cargar contenido y datos desde cualquier dominio.

Controles de permiso

El modelo de seguridad de tiempo de ejecución del cliente Flash Player se ha diseñado en torno a los recursos, que son objetos como, por ejemplo, archivos SWF, datos locales y URL de Internet. Las *personas con un interés directo* son aquellas que poseen o utilizan esos recursos. Estas personas pueden ejercer controles (configuración de seguridad) sobre sus propios recursos y cada recurso tiene cuatro personas con un interés directo. Flash Player aplica de forma estricta una jerarquía de autoridad para estos controles, como se muestra en la siguiente ilustración:



Jerarquía de controles de seguridad

Esto significa, por ejemplo, que si un administrador restringe el acceso a un recurso, ninguna otra persona con un interés directo puede anular dicha restricción.

Controles de administrador

Un usuario administrador de un equipo, que haya iniciado sesión con derechos administrativos, puede aplicar una configuración de seguridad de Flash Player que afecte a todos los usuarios del equipo. En un entorno no empresarial, como puede ser un equipo doméstico, suele haber un usuario que tiene además acceso administrativo. Incluso en un entorno empresarial, los usuarios individuales pueden tener derechos administrativos en el equipo.

Hay dos tipos de controles de usuario administrador:

- El archivo mms.cfg
- El directorio Global Flash Player Trust

El archivo mms.cfg

El archivo mms.cfg es un archivo de texto que permite a los administradores conceder o restringir el acceso a numerosas capacidades. Cuando se inicia Flash Player, lee la configuración de seguridad de este archivo y la utiliza para limitar la funcionalidad. El archivo mms.cfg incluye valores de configuración que el administrador utiliza para gestionar capacidades como, por ejemplo, controles de privacidad, seguridad de archivos locales, conexiones de socket, etc.

Un archivo SWF puede acceder a información sobre las capacidades desactivadas, mediante una llamada a las propiedades `Capabilities.avHardwareDisable` y `Capabilities.localFileReadDisable`. Sin embargo, la mayor parte de la configuración del archivo mms.cfg no puede consultarse desde ActionScript.

Para aplicar las políticas de privacidad y seguridad independientes de la aplicación en un equipo, el archivo mms.cfg sólo deben modificarlo los administradores del sistema. Los instaladores de aplicación no deben utilizar el archivo mms.cfg. Aunque un instalador que se ejecutara con privilegios de administrador podría modificar el contenido del archivo mms.cfg, Adobe considera que dicho uso infringe la confianza del usuario e insta a los creadores de los instaladores que no modifiquen nunca el archivo mms.cfg.

El archivo mms.cfg se almacena en la ubicación siguiente:

- Windows: *system*\Macromed\Flash\mms.cfg

  (por ejemplo, C:\WINDOWS\system32\Macromed\Flash\mms.cfg)

- Mac: *app support*/Macromedia/mms.cfg

  (por ejemplo, /Library/Application Support/Macromedia/mms.cfg)

Para obtener más información sobre el archivo mms.cfg, consulte la guía de administración de Flash Player www.adobe.com/go/flash_player_admin_es.

El directorio Global Flash Player Trust

Los usuarios con derechos administrativos y los instaladores pueden registrar archivos locales SWF específicos como de confianza para todos los usuarios. Estos archivos SWF se asignan al entorno limitado local de confianza. Pueden interactuar con cualquier otro archivo SWF y pueden cargar datos de cualquier ubicación local o remota. Los archivos se designan como de confianza en el directorio Global Flash Player Trust, en la ubicación siguiente:

- Windows: *system*\Macromed\Flash\FlashPlayerTrust

  (por ejemplo, C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust)

- Mac: *app support*/Macromedia/FlashPlayerTrust

  (por ejemplo, /Library/Application Support/Macromedia/FlashPlayerTrust)

El directorio Flash Player Trust puede contener cualquier número de archivos de texto, cada uno de los cuales contiene listas de rutas de confianza, con una ruta por cada línea. Cada ruta puede ser un archivo SWF individual, un archivo HTML o un directorio. Las líneas de comentario empiezan por el símbolo `#`. Por ejemplo, un archivo de configuración de confianza de Flash Player que contenga el siguiente texto concede el estado "de confianza" a todos los archivos en el directorio especificado y en todos sus subdirectorios:

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

Las rutas incluidas en un archivo de configuración de confianza deben ser siempre rutas locales o rutas de red SMB. Las rutas HTTP incluidas en un archivo de configuración de confianza se omiten; sólo los archivos locales pueden ser de confianza.

Para evitar conflictos, debe asignarse a cada archivo de configuración de confianza un nombre de archivo correspondiente a la aplicación de instalación y utilizar una extensión de archivo .cfg.

Un desarrollador que distribuye un archivo SWF ejecutado localmente a través de un instalador puede hacer que el instalador añada un archivo de configuración al directorio Global Flash Player Trust, concediendo así privilegios completos al archivo que distribuye. El instalador debe ejecutarse como usuario con derechos administrativos. A diferencia del archivo mms.cfg, el directorio Global Flash Player Trust se incluye para que los instaladores puedan conceder permisos de confianza. Tanto los usuarios administradores como los instaladores pueden designar aplicaciones locales de confianza a través del directorio Global Flash Player Trust.

Hay también directorios Flash Player Trust para usuarios individuales (consulte "Controles de usuario" en la página 720).

Controles de usuario

Flash Player ofrece tres mecanismos de nivel de usuario diferentes para definir permisos: la interfaz de usuario de configuración y el Administrador de configuración, y el directorio User Flash Player Trust.

La interfaz de usuario Configuración y el Administrador de configuración

La interfaz de usuario Configuración es un mecanismo rápido e interactivo para establecer la configuración de un determinado dominio. El Administrador de configuración presenta una interfaz más detallada y permite realizar cambios globales que afectan a los permisos de muchos de los dominios o de todos ellos. Además, cuando un archivo SWF solicita un nuevo permiso que requiere tomar decisiones en tiempo de ejecución que afectan a la seguridad o privacidad, se muestran cuadros de diálogo en los que los usuarios pueden ajustar algunos parámetros de configuración de Flash Player.

La interfaz de usuario Configuración y el Administrador de configuración proporcionan opciones relacionadas con la seguridad como, por ejemplo, configuración de micrófono y cámara y de almacenamiento de objetos compartidos, configuración relacionada con el contenido heredado, etc.

Nota: *las configuraciones establecidas en el archivo mms.cfg (consulte "Controles de administrador" en la página 718) no se reflejan en el Administrador de configuración.*

Para obtener información detallada sobre el Administrador de configuración, consulte www.adobe.com/go/settingsmanager_es.

El directorio User Flash Player Trust

Los usuarios y los instaladores pueden registrar determinados archivos SWF locales como archivos de confianza. Estos archivos SWF se asignan al entorno limitado local de confianza. Pueden interactuar con cualquier otro archivo SWF y pueden cargar datos de cualquier ubicación local o remota. Un usuario designa un archivo como archivo de confianza en el directorio User Flash Player Trust, que es el mismo directorio donde se almacenan los objetos compartidos, en las siguientes ubicaciones (las ubicaciones son específicas del usuario actual):

- Windows: app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (por ejemplo, C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust en Windows XP o C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust en Windows Vista)

  En Windows, la carpeta Datos de programa está oculta de forma predeterminada. Para mostrar carpetas y archivos ocultos, seleccione Mi PC para abrir el Explorador de Windows, seleccione Herramientas > Opciones de carpeta y, a continuación, la ficha Ver. En la ficha Ver, seleccione el botón de opción Mostrar todos los archivos y carpetas ocultos.

- Mac: app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (por ejemplo, /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

Esta configuración sólo afecta al usuario actual, no a los demás usuarios que inician sesión en el equipo. Si un usuario sin derechos administrativos instala una aplicación en su parte del sistema, el directorio User Flash Player Trust permite al instalador registrar la aplicación como "de confianza" para dicho usuario.

Un desarrollador que distribuye un archivo SWF ejecutado localmente a través de un instalador puede hacer que el instalador añada un archivo de configuración al directorio User Flash Player Trust, concediendo así privilegios completos al archivo que distribuye. Incluso en esta situación, el archivo del directorio User Flash Player Trust se considera un control de usuario, porque se inicia como consecuencia de una acción del usuario (la instalación).

También hay un directorio Global Flash Player Trust que el usuario administrador o el instalador utiliza para registrar una aplicación para todos los usuarios de un equipo (consulte “Controles de administrador” en la página 718).

Controles de sitio Web (archivos de política)

Para que los datos de su servidor Web estén disponibles para los archivos SWF de otros dominios, se puede crear un archivo de política en el servidor. Un *archivo de política* es un archivo XML situado en una ubicación específica de su servidor.

Los archivos de política afectan a varios activos, incluidos los siguientes:

- Datos en mapas de bits, sonidos y vídeos
- Carga de archivos de texto y XML
- Importación de archivos SWF desde otros dominios de seguridad en el dominio de seguridad del archivo SWF que realiza la carga
- Acceso a conexiones de socket y conexiones de socket XML

Los objetos ActionScript crean una instancia de dos tipos diferentes de conexiones de servidor: conexiones de servidor basadas en documentos y conexiones de socket. Los objetos de ActionScript como Loader, Sound, URLLoader y URLStream crean instancias de conexiones de servidor basadas en documentos y estos objetos cargan un archivo de una URL. Los objetos Socket y XMLSocket de ActionScript realizan conexiones de socket, que funcionan con datos de transmisión y no con documentos cargados.

Debido a que Flash Player admite dos tipos de conexiones de servidor, existen dos tipos de archivos de política: archivos de política URL y archivos de política de socket.

- Las conexiones basadas en documentos requieren *archivos de política URL*. Estos archivos permiten al servidor indicar que sus datos y documentos están disponibles para los archivos SWF de dominios determinados o de todos los dominios.
- Las conexiones de socket requieren *archivos de política de socket*, que permiten establecer redes directamente en el nivel inferior de socket TCP a través de las clases Socket y XMLSocket.

Flash Player requiere que los archivos de política se transmitan a través del mismo protocolo que pretende utilizar la conexión que se desea establecer. Por ejemplo, cuando se incluye un archivo de política en el servidor HTTP, los archivos SWF de otros dominios pueden cargar datos de él como un servidor HTTP. Sin embargo, si no se proporciona un archivo de política de socket en el mismo servidor, se prohibirá a los archivos SWF de otros dominios que se conecten con el servidor en el socket. Dicho de otro modo, la vía por la cual se recupera un archivo de política debe coincidir con la vía empleada en la conexión.

En el resto de esta sección se describen brevemente el uso y la sintaxis de los archivos de política, ya que se aplican a los archivos SWF publicados para Flash Player 10. (La implementación de archivos de política es ligeramente diferente en las versiones anteriores de Flash Player, ya que las versiones posteriores han reforzado la seguridad del programa.) Para obtener información más detallada sobre los archivos de política, consulte el tema sobre los cambios de los archivos de política en Flash Player 9 del Flash Player Developer Center (centro de desarrolladores de Flash Player) en la dirección www.adobe.com/go/devnet_security_es.

Archivos maestros de política

De forma predeterminada, Flash Player (y el contenido de AIR que no está en el entorno limitado de la aplicación AIR) busca un archivo de política URL denominado `crossdomain.xml` en el directorio raíz del servidor y un archivo de política de socket en el puerto 843. Los archivos que se encuentran en cualquiera de estas dos ubicaciones se denominan *archivos maestros de política*. (En el caso de las conexiones de socket, Flash Player también busca un archivo de política de socket en el mismo puerto que utiliza la conexión principal. No obstante, los archivos de política que se encuentran en dicho puerto no se consideran archivos maestros de política.)

Además de especificar los permisos de acceso, el archivo maestro de política puede contener también una sentencia *meta-policy*. Una metapolítica especifica las ubicaciones que pueden contener archivos de política. La metapolítica predeterminada de los archivos de política URL es "master-only", es decir, /crossdomain.xml es el único archivo de política permitido en el servidor. La metapolítica predeterminada de los archivos de política de socket es "all", es decir, cualquier socket del host puede servir un archivo de política de socket.

Nota: *en Flash Player 9 y versiones anteriores, la metapolítica predeterminada de los archivos de política URL era "all", es decir, cualquier directorio podía contener un archivo de política. Si ha implementado aplicaciones que cargan archivos de política desde ubicaciones diferentes al archivo predeterminado /crossdomain.xml, y dichas aplicaciones pudieran ejecutarse en Flash Player 10, asegúrese de que usted (o el administrador del servidor) modifica el archivo maestro de política para permitir archivos de política adicionales. Para obtener más información sobre cómo especificar una metapolítica diferente, consulte el tema sobre los cambios de los archivos de política en Flash Player 9 del Flash Player Developer Center (centro de desarrolladores de Flash Player) en la dirección www.adobe.com/go/devnet_security_es.*

Un archivo SWF puede comprobar un nombre de archivo de política o una ubicación de directorio diferentes llamando al método `Security.loadPolicyFile()`. Sin embargo, si el archivo maestro de política no especifica que la ubicación de destino puede servir archivos de política, la llamada a `loadPolicyFile()` no tendrá efecto, incluso si hay un archivo de política en dicha ubicación. Llame a `loadPolicyFile()` antes de intentar cualquier operación de red que requiera el archivo de política. Flash Player pone en cola automáticamente las peticiones de red detrás de sus correspondientes intentos de archivo de política. De este modo, es aceptable, por ejemplo, llamar a `Security.loadPolicyFile()` inmediatamente antes de iniciar una operación de red.

Al comprobar un archivo maestro de política, Flash Player espera durante tres segundos una respuesta del servidor. Si no recibe ninguna respuesta, Flash Player asumirá que no existe ningún archivo maestro de política. Sin embargo, no hay un valor de tiempo de espera predeterminado para las llamadas a `loadPolicyFile()`; Flash Player asume que el archivo al que se llama existe, y esperará el tiempo necesario para cargarlo. Por tanto, si desea asegurarse de que se carga un archivo maestro de política, utilice `loadPolicyFile()` para cargarlo explícitamente.

Aunque el método se denomina `Security.loadPolicyFile()`, no se cargará ningún archivo de política hasta que no se realice una llamada de red que requiera un archivo de política. Las llamadas a `loadPolicyFile()` sólo indican a Flash Player la ubicación en la que buscar archivos de política cuando éstos se necesitan.

No puede recibir una notificación del momento en el que se inicia o completa una solicitud de archivo de política, y no hay razón para ello. Flash Player realiza comprobaciones de política de forma asincrónica, y espera automáticamente a iniciar las conexiones hasta que las comprobaciones de archivos de política se hayan realizado correctamente.

Las secciones siguientes incluyen información que se aplica sólo a los archivos de política URL. Para obtener más información sobre los archivos de política de socket, consulte "Conexión a sockets" en la página 737.

Ámbito de los archivos de política URL

Un archivo de política URL sólo se aplica al directorio desde el cual se carga y a sus directorios secundarios. Un archivo de política en el directorio raíz se aplica a todo el servidor. Un archivo de política cargado desde un subdirectorio arbitrario sólo se aplica a dicho directorio y a sus subdirectorios.

Un archivo de política sólo afecta al acceso al servidor concreto en el que reside. Por ejemplo, un archivo de política ubicado en https://www.adobe.com:8080/crossdomain.xml sólo se aplica a las llamadas para cargar datos realizadas a www.adobe.com en el puerto 8080 utilizando el protocolo HTTPS.

Especificación de los permisos de acceso en un archivo de política URL

Un archivo de política contiene una sola etiqueta `<cross-domain-policy>`, la cual no contiene ninguna o contiene varias etiquetas `<allow-access-from>`. Cada etiqueta `<allow-access-from>` contiene un atributo, `domain`, el cual especifica una dirección IP exacta, un dominio exacto o un dominio comodín (cualquier dominio). Los dominios comodín se indican de una de las dos formas siguientes:

- Mediante un solo asterisco (*), que coincide con todos los dominios y todas las direcciones IP
- Mediante un asterisco seguido de un sufijo, que coincide sólo con los dominios que terminan con el sufijo especificado

Los sufijos deben empezar por un punto. Sin embargo, los dominios comodín con sufijos pueden incluir los dominios formados únicamente por el sufijo sin el punto inicial. Por ejemplo, xyz.com se considera parte de *.xyz.com. Los comodines no se pueden utilizar en las especificaciones de dominio IP.

En el ejemplo siguiente se muestra un archivo de política URL que permite el acceso a archivos SWF procedentes de *.example.com, www.friendOfExample.com y 192.0.34.166:

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*.example.com" />
    <allow-access-from domain="www.friendOfExample.com" />
    <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

Si especifica una dirección IP, sólo se otorgará acceso a los archivos SWF cargados desde esa dirección IP mediante la sintaxis de IP (por ejemplo, http://65.57.83.12/flashmovie.swf). No se concederá acceso a los archivos SWF a través de la sintaxis de nombre de dominio. Flash Player no lleva a cabo la resolución DNS.

Se puede permitir el acceso a documentos procedentes de cualquier dominio, tal y como se muestra en el siguiente ejemplo:

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

Cada etiqueta `<allow-access-from>` tiene además el atributo opcional `secure`, que tiene el valor predeterminado `true`. Si su archivo de política se encuentra en un servidor HTTPS y desea permitir que los archivos SWF de un servidor que no sea HTTPS puedan cargar datos del servidor HTTPS, puede establecer el atributo en `false`.

Establecer el atributo `secure` en `false` puede poner en peligro la seguridad proporcionada por HTTPS. En concreto, establecer este atributo en `false` deja el contenido seguro expuesto a posibles ataques de fisgones y suplantadores. Adobe recomienda encarecidamente no establecer el atributo `secure` en `false`.

Si los datos que se van a cargar se encuentran en un servidor HTTPS, pero el archivo SWF que los carga se encuentra en un servidor HTTP, Adobe recomienda mover el archivo SWF de carga a un servidor HTTPS. De este modo, se podrán mantener todas las copias de sus datos seguros bajo la protección de HTTPS. Sin embargo, si se decide mantener el archivo SWF que realiza la carga en un servidor HTTP, deberá añadirse el atributo `secure="false"` a la etiqueta `<allow-access-from>`, como se muestra en el código siguiente:

```
<allow-access-from domain="www.example.com" secure="false" />
```

Otro elemento que se puede utilizar para permitir el acceso es la etiqueta `allow-http-request-headers-from`. Este elemento permite a un cliente que aloja contenido de otro dominio de permisos enviar a su dominio encabezados definidos por el usuario. Mientras que la etiqueta `<allow-access-from>` concede permiso a otros dominios para extraer datos de su dominio, la etiqueta `allow-http-request-headers-from` concede permiso a otros dominios para introducir datos en el mismo, en la forma de encabezados. En el ejemplo siguiente, se permite a cualquier dominio enviar el encabezado SOAPAction al dominio actual:

```
<cross-domain-policy>
    <allow-http-request-headers-from domain="*" headers="SOAPAction"/>
</cross-domain-policy>
```

Si la sentencia `allow-http-request-headers-from` se encuentra en el archivo maestro de política, se aplicará a todos los directorios del host. De lo contrario, sólo se aplicará al directorio y los subdirectorios del archivo de política que contiene la sentencia.

Precarga de archivos de política

La carga de datos de un servidor o la conexión a un socket es una operación asincrónica. Flash Player simplemente espera a que el archivo de política acabe de descargarse para comenzar la operación principal. Sin embargo, la extracción de datos de píxeles de imágenes o la extracción de datos de ejemplo de sonidos es una operación sincrónica. Para poder extraer datos, primero se debe cargar el archivo de política. Al cargar un medio, especifique que busque un archivo de política:

- Si utiliza el método `Loader.load()`, establezca la propiedad `checkPolicyFile` del parámetro `context`, que es un objeto LoaderContext.
- Si incorpora una imagen en un campo de texto mediante la etiqueta `<img>`, establezca el atributo `checkPolicyFile` de la etiqueta `<img>` en `"true"`, como en el siguiente ejemplo:

  ```
  <img checkPolicyFile = "true" src = "example.jpg">
  ```

- Si utiliza el método `Sound.load()`, establezca la propiedad `checkPolicyFile` del parámetro `context`, que es un objeto SoundLoaderContext.
- Si utiliza la clase NetStream, establezca la propiedad `checkPolicyFile` del objeto NetStream.

Si se establece uno de estos parámetros, Flash Player comprueba primero si hay algún archivo de política que ya se haya descargado para dicho dominio. A continuación, busca el archivo de política en la ubicación predeterminada del servidor, comprobando las sentencias `<allow-access-from>` y la presencia de una metapolítica. Por último, considera las llamadas pendientes al método `Security.loadPolicyFile()` para comprobar si se encuentran dentro del ámbito correspondiente.

Controles de autor (desarrollador)

La principal API de ActionScript que se utiliza para conceder privilegios de seguridad es el método `Security.allowDomain()`, que concede privilegios a archivos SWF en los dominios especificados. En el siguiente ejemplo, un archivo SWF concede acceso a archivos SWF que se encuentran disponibles en el dominio www.example.com:

```
Security.allowDomain("www.example.com")
```

Este método concede permisos para lo siguiente:

- Reutilización de scripts entre archivos SWF (consulte "Reutilización de scripts" en la página 731)
- Acceso a la lista de visualización (consulte "Recorrido de la lista de visualización" en la página 734)
- Detección de eventos (consulte "Seguridad de eventos" en la página 734)
- Acceso completo a las propiedades y métodos del objeto Stage (consulte "Seguridad del objeto Stage" en la página 733)

El principal objetivo de llamar al método `Security.allowDomain()` es conceder permiso para que los archivos SWF de un dominio exterior puedan manipular mediante script el archivo SWF, a través de una llamada al método `Security.allowDomain()`. Para obtener más información, consulte "Reutilización de scripts" en la página 731.

Cuando se especifica una dirección IP como parámetro en el método `Security.allowDomain()`, no se permite el acceso de todas las partes que tienen su origen en la dirección IP especificada. Sólo se permite el acceso de la parte que contiene la dirección IP especificada en su URL, y no un nombre de dominio que corresponda a la dirección IP. Por ejemplo, si el nombre de dominio www.example.com corresponde a la dirección IP 192.0.34.166, una llamada a `Security.allowDomain("192.0.34.166")` no concede el acceso a www.example.com.

Se puede pasar el comodín `"*"` al método `Security.allowDomain()` para permitir el acceso desde todos los dominios. Como se concede permiso a los archivos SWF de *todos* los dominios para que manipulen mediante script el archivo SWF que realiza la llamada, se debe utilizar el comodín `"*"` con precaución.

ActionScript incluye una segunda API de permiso, denominada `Security.allowInsecureDomain()`. Este método realiza lo mismo que el método `Security.allowDomain()` y además, cuando se llama desde un archivo SWF que se encuentra disponible en una conexión HTTPS segura, permite el acceso al archivo SWF que realiza la llamada por parte de otros archivos SWF que se encuentran disponibles en un protocolo no seguro, como HTTP. Sin embargo, no es seguro permitir la reutilización de scripts entre archivos desde un protocolo seguro (HTTPS) y desde protocolos no seguros (como HTTP); si se permite, se deja el contenido seguro expuesto a posibles ataques de fisgones y suplantadores. Este es el modo en que pueden funcionar estos ataques: como el método `Security.allowInsecureDomain()` permite que los archivos SWF proporcionados a través de conexiones HTTP puedan acceder a los datos de HTTPS seguro, un atacante interpuesto entre el servidor HTTP y los usuarios podría sustituir el archivo SWF de HTTP por uno propio y acceder así a los datos HTTPS.

Otro método importante relacionado con la seguridad es el método `Security.loadPolicyFile()`, que hace que Flash Player compruebe si hay un archivo de política en una ubicación no estándar. Para obtener más información, consulte "Controles de sitio Web (archivos de política)" en la página 721.

Restricción de las API de red

Las API de red se pueden restringir de dos formas. Para evitar actividad malintencionada, se bloquea el acceso a los puertos que suelen estar reservados; estos bloques no se pueden sustituir en el código. Para controlar el acceso de un archivo SWF a la funcionalidad de red con respecto a otros puertos, puede utilizar la configuración `allowNetworking`.

Puertos bloqueados

Flash Player y Adobe AIR, al igual que los navegadores, tienen restricciones para el acceso HTTP a determinados puertos. No se permiten las peticiones HTTP a determinados puertos estándar que se utilizan tradicionalmente para tipos de servidores que no son HTTP.

Las API que acceden a una URL de red están sujetas a estas restricciones de bloqueo de puertos. La única excepción son las API que llaman directamente a sockets, como `Socket.connect()` y `XMLSocket.connect()`, o las llamadas a `Security.loadPolicyFile()`, donde se carga un archivo de política de socket. Las conexiones de socket se permiten o deniegan a través del uso de archivos de política de socket en el servidor de destino.

La lista siguiente muestra las API de ActionScript 3.0 donde se aplica el bloqueo de puertos:

`FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `Loader.loadBytes()`, `navigateToURL()`, `NetConnection.call()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `sendToURL()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`

El bloqueo de puertos también se aplica a la importación de biblioteca compartida, el uso de la etiqueta `<img>` en campos de texto y la carga de archivos SWF en páginas HTML a través de las etiquetas `<object>` y `<embed>`.

Las listas siguientes muestran los puertos que están bloqueados:

HTTP: 20 (datos ftp), 21 (control ftp)

HTTP y FTP: 1 (tcpmux), 7 (echo), 9 (discard), 11 (systat), 13 (daytime), 15 (netstat), 17 (qotd), 19 (chargen), 22 (ssh), 23 (telnet), 25 (smtp), 37 (time), 42 (name), 43 (nicname), 53 (domain), 77 (priv-rjs), 79 (finger), 87 (ttylink), 95 (supdup), 101 (hostriame), 102 (iso-tsap), 103 (gppitnp), 104 (acr-nema), 109 (pop2), 110 (pop3), 111 (sunrpc), 113 (auth), 115 (sftp), 117 (uucp-path), 119 (nntp), 123 (ntp), 135 (loc-srv / epmap), 139 (netbios), 143 (imap2), 179 (bgp), 389 (ldap), 465 (smtp+ssl), 512 (print / exec), 513 (login), 514 (shell), 515 (printer), 526 (tempo), 530 (courier), 531 (chat), 532 (netnews), 540 (uucp), 556 (remotefs), 563 (nntp+ssl), 587 (smtp), 601 (syslog), 636 (ldap+ssl), 993 (ldap+ssl), 995 (pop3+ssl), 2049 (nfs), 4045 (lockd), 6000 (x11)

Utilización del parámetro allowNetworking

Puede controlar el acceso de un archivo SWF a la funcionalidad de red configurando el parámetro `allowNetworking` en las etiquetas `<object>` y `<embed>` de la página HTML que aloja el contenido SWF.

Los valores posibles de `allowNetworking` son:

- `"all"` (valor predeterminado): se admiten todas las API de red en el archivo SWF.
- `"internal"`: el archivo SWF no puede llamar a las API de navegación o de interacción con el navegador que se muestran a continuación, pero puede llamar a las otras API de red.
- `"none"`: el archivo SWF no puede llamar a ninguna de las API de navegación o de interacción con el navegador que se muestran a continuación, y no puede usar ninguna de las API de comunicación entre archivos SWF, incluidas también en la lista siguiente.

El parámetro `allowNetworking` se ha diseñado principalmente para los casos en los que el archivo SWF y la página HTML que lo incluye pertenecen a dominios diferentes. No se recomienda el uso del valor de `"internal"` ni `"none"` cuando el archivo SWF que se carga pertenece al mismo dominio que las páginas HTML que lo incluyen, porque no se puede garantizar que un archivo SWF se cargue siempre con la página HTML deseada. Las partes que no son de confianza podrían cargar un archivo SWF de su dominio sin ninguna página HTML que lo incluya, en cuyo caso la restricción `allowNetworking` no funcionará del modo esperado.

Al llamar a una API no permitida, se emite una excepción SecurityError.

Añada el parámetro `allowNetworking` y establezca su valor en las etiquetas `<object>` y `<embed>` de la página HTML que contiene una referencia al archivo SWF, como se muestra en el ejemplo siguiente:

```
<object classic="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    Code
base="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,
0"
    width="600" height="400" ID="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Una página HTML también puede utilizar un script para generar etiquetas que incorporan SWF. Necesita alterar el script para que inserte el valor `allowNetworking` adecuado. Las páginas HTML generadas por Flash y Adobe Flex Builder utilizan la función `AC_FL_RunContent()` para incorporar referencias a archivos SWF. Añada la configuración del parámetro `allowNetworking` al script, como en el siguiente ejemplo:

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

Las siguientes API no se admiten cuando `allowNetworking` está establecido en `"internal"`:

`navigateToURL()`, `fscommand()`, `ExternalInterface.call()`

Además de las API anteriores, no se admiten las siguientes API cuando `allowNetworking` se establece en `"none"`:

`sendToURL()`, `FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `LocalConnection.connect()`, `LocalConnection.send()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `SharedObject.getLocal()`, `SharedObject.getRemote()`, `Socket.connect()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`, `XMLSocket.connect()`

Aunque la configuración de `allowNetworking` seleccionada permita a un archivo SWF usar una API de red, puede haber otras restricciones basadas en las limitaciones de un entorno limitado de seguridad (consulte “Entornos limitados de seguridad” en la página 716).

Cuando `allowNetworking` se establece en `"none"`, no se puede hacer referencia a medios externos en una etiqueta `<img>` en la propiedad `htmlText` de un objeto TextField (se emite una excepción SecurityError).

Si se establece `allowNetworking` en `"none"`, un símbolo de una biblioteca compartida importada añadido en la herramienta de edición Flash (no mediante ActionScript) se bloqueará en tiempo de ejecución.

Seguridad del modo de pantalla completa

En Player 9.0.27.0 y versiones posteriores se admite el modo de pantalla completa, en el que el contenido que se ejecuta en Flash Player puede llenar toda la pantalla. Para entrar en el modo de pantalla completa, se establece la constante `StageDisplayState.FULL_SCREEN` como valor de la propiedad `displayState` de Stage. Para obtener más información, consulte “Trabajo con el modo de pantalla completa” en la página 294.

Hay que tener en cuenta algunas consideraciones de seguridad relacionadas con los archivos SWF que se ejecutan en un navegador.

Para activar el modo de pantalla completa, en las etiquetas `<object>` y `<embed>` de la página HTML que contiene una referencia al archivo SWF, añada el parámetro `allowFullScreen`, con su valor establecido en `"true"` (el valor predeterminado es `"false"`), tal y como se muestra en el siguiente ejemplo:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"

codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,
18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Una página HTML también puede utilizar un script para generar etiquetas que incorporan SWF. Es necesario modificar el script para insertar la configuración adecuada de `allowFullScreen`. Las páginas HTML generadas por Flash y Flex Builder utilizan la función `AC_FL_RunContent()` para incorporar referencias a archivos SWF y es necesario añadir la configuración del parámetro `allowFullScreen` al script, como en el siguiente ejemplo:

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

El código ActionScript que inicia el modo de pantalla completa sólo puede llamarse como respuesta a un evento de ratón o de teclado. Si se llama en otras situaciones, Flash Player emite una excepción.

Cuando se entra en el modo de pantalla completa, aparece un mensaje que indica al usuario cómo puede salir y volver al modo normal. El mensaje aparece durante unos segundos y luego desaparece progresivamente.

Para el contenido que se está ejecutando en un navegador, el uso del teclado se limita al modo de pantalla completa. En Flash Player 9, sólo se admiten los métodos abreviados de teclado que devuelven la aplicación a modo normal como, por ejemplo, presionar la tecla Esc. Los usuarios no pueden introducir texto en los campos de texto ni desplazarse por la pantalla. En Flash Player 10 y versiones posteriores, se admiten determinadas teclas que no afectan a la impresión (concretamente, las teclas de flecha, la barra espaciadora y el tabulador). Sin embargo, la entrada de texto aún está prohibida.

El modo de pantalla completa siempre se permite en el reproductor autónomo o en un archivo de proyector. Asimismo, el uso del teclado (incluyendo la entrada de texto) es totalmente compatible en estos entornos.

Al llamar a la propiedad `displayState` de un objeto Stage, se emite una excepción para cualquier origen de llamada que no esté en el mismo entorno limitado de seguridad que el propietario del objeto Stage (el archivo SWF principal). Para obtener más información, consulte "Seguridad del objeto Stage" en la página 733.

Los administradores pueden desactivar el modo de pantalla completa en los archivos SWF que se ejecutan en navegadores. Para ello, deben establecer `FullScreenDisable = 1` en el archivo mms.cfg. Para obtener información más detallada, consulte "Controles de administrador" en la página 718.

En un navegador, un archivo SWF debe estar contenido en una página HTML para poder verlo en el modo de pantalla completa.

Carga de contenido

Un archivo SWF puede cargar los siguientes tipos de contenido:

- Archivos SWF
- Imágenes
- Sonido
- Vídeo

Carga de archivos SWF e imágenes

La clase Loader se utiliza para cargar archivos SWF e imágenes (archivos JPG, GIF o PNG). Cualquier archivo SWF que no se encuentre en el entorno limitado local con sistema de archivos puede cargar archivos SWF e imágenes desde cualquier dominio de red. Sólo los archivos SWF de entornos limitados locales pueden cargar archivos SWF e imágenes del sistema de archivos local. Sin embargo, los archivos del entorno limitado local con acceso a la red sólo pueden cargar archivos SWF que se encuentren en el entorno limitado local de confianza o en el entorno limitado local con acceso a la red. Los archivos SWF del entorno limitado local con acceso a la red cargan contenido local que no sean archivos SWF (por ejemplo, imágenes), pero no pueden acceder a los datos del contenido cargado.

Al cargar un archivo SWF de un origen que no es de confianza (por ejemplo, un dominio distinto al del archivo SWF raíz del objeto Loader), es aconsejable definir una máscara para el objeto Loader para evitar que el contenido cargado (que es un elemento secundario del objeto Loader) se dibuje en partes del escenario situadas fuera de la máscara, como se muestra en el siguiente código:

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

Cuando se llama al método `load()` del objeto Loader, se puede especificar un parámetro `context`, que es un objeto LoaderContext. La clase LoaderContext incluye tres propiedades que permiten definir el contexto de uso del contenido cargado:

- `checkPolicyFile`: utilice esta propiedad sólo para cargar un archivo de imagen (no un archivo SWF). Esta propiedad se especifica en un archivo de imagen de un dominio ajeno al del archivo que contiene el objeto Loader. Si establece esta propiedad en `true`, Loader comprueba el servidor de origen de un archivo de política URL (consulte "Controles de sitio Web (archivos de política)" en la página 721). Si el servidor concede permiso al dominio de Loader, el código ActionScript de los archivos SWF del dominio de Loader puede acceder a los datos de la imagen cargada. Dicho de otro modo, se puede utilizar la propiedad `Loader.content` para obtener una referencia al objeto Bitmap que representa la imagen cargada o el método `BitmapData.draw()` para acceder a los píxeles de la imagen cargada.

- `securityDomain`: utilice esta propiedad sólo si carga un archivo SWF (no una imagen). Esta propiedad se especifica en un archivo SWF de un dominio ajeno al del archivo que contiene el objeto Loader. Actualmente, la propiedad `securityDomain` sólo admite dos valores: `null` (el valor predeterminado) y `SecurityDomain.currentDomain`. Si se especifica `SecurityDomain.currentDomain`, el archivo SWF cargado debe *importarse* en el entorno limitado del objeto SWF que realiza la carga, lo que significa que funciona como si se hubiera cargado del servidor del archivo SWF que realiza la carga. Esto sólo se permite si se encuentra un archivo de política URL en el servidor del archivo SWF cargado que permita el acceso por parte del dominio del archivo SWF que realiza la carga. Si se encuentra el archivo de política necesario, el cargador y el contenido cargado pueden manipularse mutuamente mediante script en cuanto se inicia la carga, ya que se encuentran en el mismo entorno limitado. El entorno limitado donde se importa el archivo puede sustituirse a través de una carga normal, seguida de una llamada del archivo SWF cargado al método `Security.allowDomain()`. Es posible que este último método sea más sencillo, pues el archivo SWF cargado estará entonces en su entorno limitado natural y, por lo tanto, podrá acceder a los recursos de su propio servidor real.
- `applicationDomain`: utilice esta propiedad solamente si carga un archivo SWF escrito en ActionScript 3.0 (no una imagen ni un archivo SWF escritos en ActionScript 1.0 ó 2.0). Al cargar el archivo, puede especificar que se incluya el archivo en un dominio de aplicación concreto y no en un nuevo dominio de aplicación que sea un elemento secundario del dominio de aplicación del archivo SWF que realiza la carga, que es lo que sucede de forma predeterminada. Tenga en cuenta que los dominios de aplicación son subunidades de los dominios de seguridad y, por lo tanto, puede especificar un dominio de aplicación de destino únicamente si el archivo SWF que está cargando procede de su propio dominio de seguridad, ya sea porque corresponde a su servidor propio o porque lo ha importado en su dominio de seguridad a través de la propiedad `securityDomain`. Si especifica un dominio de aplicación pero el archivo SWF cargado forma parte de un dominio de seguridad distinto, el dominio especificado en `applicationDomain` se omite. Para obtener más información, consulte "Utilización de la clase ApplicationDomain" en la página 667.

Para obtener más detalles, consulte "Especificación del contexto de carga" en la página 322.

Una propiedad importante de un objeto Loader es `contentLoaderInfo`, que es un objeto LoaderInfo. A diferencia de lo que ocurre con la mayoría de los objetos, un objeto LoaderInfo se comparte entre el archivo SWF que realiza la carga y el contenido cargado, y siempre es accesible para ambas partes. Cuando el contenido cargado es un archivo SWF, éste puede acceder al objeto LoaderInfo a través de la propiedad `DisplayObject.loaderInfo`. Los objetos LoaderInfo contienen información como el progreso de carga, los URL del cargador y del contenido cargado, o la relación de confianza entre ambos. Para obtener más información, consulte "Supervisión del progreso de carga" en la página 321.

Carga de sonido y vídeos

Cualquier archivo SWF que no se encuentre en el entorno limitado local con sistema de archivos puede cargar sonido y vídeo de orígenes de red, a través de los métodos `Sound.load()`, `NetConnection.connect()` y `NetStream.play()`.

Sólo los archivos SWF locales pueden cargar medios del sistema de archivos local. Sólo los archivos SWF que se encuentren en el entorno limitado local con sistema de archivos o el entorno limitado local de confianza pueden acceder a los datos de estos archivos cargados.

Hay otras restricciones relativas al acceso de datos desde medios cargados. Para obtener información más detallada, consulte "Acceso a medios cargados como datos" en la página 734.

Carga de archivos SWF e imágenes mediante la etiqueta <img> de un campo de texto

Puede cargar archivos SWF y mapas de bits en un campo de texto mediante la etiqueta `<img>`, como se muestra en el código siguiente:

```
<img src = 'filename.jpg' id = 'instanceName' >
```

Para acceder al contenido cargado de este modo, puede utilizar el método `getImageReference()` de la instancia de TextField, como se muestra en el código siguiente:

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

Sin embargo, tenga en cuenta que los archivos SWF e imágenes que se cargan de este modo se incluyen en el entorno limitado correspondiente a su origen.

Si se carga un archivo de imagen mediante una etiqueta `<img>` en un campo de texto, puede ser que haya un archivo de política URL que permita el acceso a datos de la imagen. Para comprobar si hay un archivo de política, añada un atributo `checkPolicyFile` a la etiqueta `<img>`, como se muestra en el código siguiente:

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

Si carga un archivo SWF mediante una etiqueta `<img>` en un campo de texto, puede permitir el acceso a los datos de dicho archivo SWF a través de una llamada al método `Security.allowDomain()`.

Si utiliza una etiqueta `<img>` en un campo de texto para cargar un archivo externo (en lugar de usar una clase Bitmap incorporada en el archivo SWF), se crea automáticamente un objeto Loader como elemento secundario del objeto TextField y el archivo externo se carga en dicho objeto Loader, tal y como sucedería si hubiera utilizado un objeto Loader en ActionScript para cargar el archivo. En este caso, el método `getImageReference()` devuelve el objeto Loader que se creó automáticamente. No es necesario realizar ninguna comprobación de seguridad para acceder a este objeto Loader porque se encuentra en el mismo entorno limitado de seguridad que el código que realiza la llamada.

Sin embargo, si se hace referencia a la propiedad `content` del objeto Loader para acceder al medio cargado, sí es preciso aplicar las reglas de seguridad. Si el contenido es una imagen, deberá implementar un archivo de política URL y, si es un archivo SWF, deberá hacer que el código del archivo SWF llame al método `allowDomain()`.

Contenido proporcionado a través de servidores RTMP

Flash Media Server utiliza el protocolo RTMP (Real-Time Media Protocol) para proporcionar datos, audio y vídeo. Un archivo SWF carga estos medios mediante el método `connect()` de la clase NetConnection y pasa un URL RTMP como parámetro. Flash Media Server puede restringir las conexiones y evitar la descarga del contenido, en función del dominio del archivo que realiza la solicitud. Para ver más detalles, consulte la documentación de Flash Media Server.

En medios cargados de orígenes RTMP, no se pueden utilizar los métodos `BitmapData.draw()` y `SoundMixer.computeSpectrum()` para extraer datos de sonido y gráficos de tiempo de ejecución.

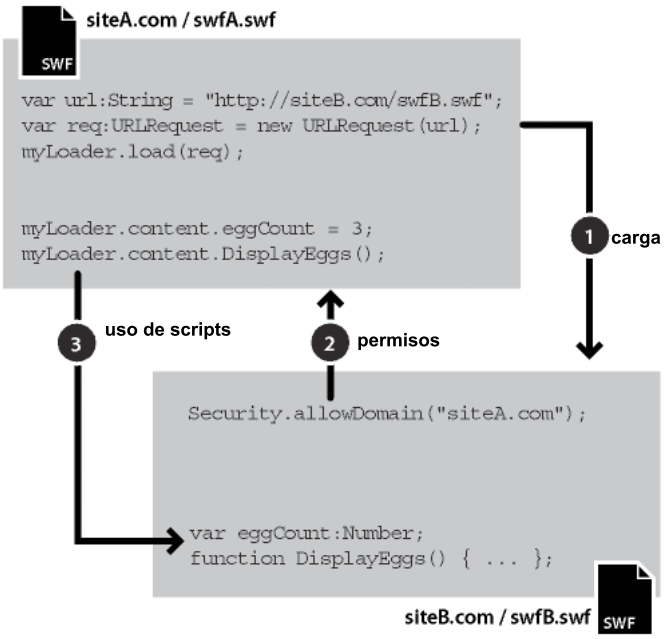
Reutilización de scripts

Si hay dos archivos SWF escritos en ActionScript 3.0 en el mismo dominio (por ejemplo, el URL de un archivo SWF es http://www.example.com/swfA.swf y el URL del otro es http://www.example.com/swfB.swf), un archivo SWF puede examinar y modificar las variables, objetos, propiedades, métodos, etc. en el otro archivo y viceversa. Esto se denomina *reutilización de scripts*.

La reutilización de scripts no se permite entre archivos SWF AVM1 y archivos SWF AVM2. Un archivo SWF AVM1 es aquél que se crea en ActionScript 1.0 o ActionScript 2.0. (AVM1 y AVM2 se refieren a la máquina virtual ActionScript.) Sin embargo, es posible utilizar la clase LocalConnection para enviar datos entre AVM1 y AVM2.

Si los dos archivos SWF escritos en ActionScript 3.0 están en dominios diferentes (por ejemplo, http://siteA.com/swfA.swf y http://siteB.com/siteB.swf), de forma predeterminada Flash Player no permite a swfA.swf usar scripts en swfB.swf, ni viceversa. Un archivo SWF concede permiso a archivos SWF de otros dominios mediante una llamada a `Security.allowDomain()`. Si se llama a `Security.allowDomain("siteA.com")`, swfB.swf concede permiso a los archivos SWF de siteA.com para utilizarlo en scripts.

En cualquier operación entre dominios es importante tener claro qué dos partes están involucradas. En esta sección, llamaremos *parte que accede* a la que lleva a cabo la reutilización de scripts (normalmente el archivo SWF que accede a otro) y *parte a la que se accede* a la otra (por lo general, el archivo SWF al que se accede). Si siteA.swf usa scripts en siteB.swf, siteA.swf será la parte que accede y siteB.swf será la parte a la que se accede, tal y como se indica en la siguiente ilustración:



Los permisos entre dominios establecidos con el método `Security.allowDomain()` son asimétricos. En el ejemplo anterior, siteA.swf puede manipular mediante script a siteB.swf, pero siteB.swf no puede hacerlo con siteA.swf, ya que siteA.swf no ha llamado al método `Security.allowDomain()` para dar permiso a los archivos SWF de siteB.com para manipularlo mediante script. Para configurar permisos simétricos, es necesario que ambos archivos SWF llamen al método `Security.allowDomain()`.

Flash Player protege los archivos SWF no sólo de los scripts creados en varios dominios por otros archivos SWF, sino también de los originados por archivos HTML. El uso de scripts de HTML en SWF puede producirse mediante funciones callback establecidas a través del método `ExternalInterface.addCallback()`. Cuando el uso de scripts de HTML en SWF traspasa los dominios, el archivo SWF debe llamar al método `Security.allowDomain()` para evitar que la operación falle, tanto si es la parte que accede como si es la parte a la que se accede. Para obtener más información, consulte "Controles de autor (desarrollador)" en la página 725.

Además, Flash Player proporciona controles de seguridad para el uso de scripts de SWF en HTML. Para obtener más información, consulte "Control del acceso URL saliente" en la página 741.

Seguridad del objeto Stage

Algunas propiedades y métodos del objeto Stage sólo están disponibles para los objetos Sprite o MovieClip de la lista de visualización.

No obstante, se dice que el objeto Stage tiene un propietario: el primer archivo SWF cargado. De forma predeterminada, las siguientes propiedades y métodos del objeto Stage sólo están disponibles para los archivos SWF que se encuentran en el mismo entorno limitado de seguridad que el propietario de Stage:

Propiedades	Métodos
`align`	`addChild()`
displayState	`addChildAt()`
frameRate	`addEventListener()`
`height`	`dispatchEvent()`
`mouseChildren`	`hasEventListener()`
`numChildren`	`setChildIndex()`
`quality`	`willTrigger()`
`scaleMode`	
`showDefaultContextMenu`	
`stageFocusRect`	
`stageHeight`	
`stageWidth`	
`tabChildren`	
`textSnapshot`	
`width`	

Para que un archivo SWF de un entorno limitado ajeno al del propietario de Stage pueda acceder a estas propiedades y métodos, el archivo SWF del propietario de Stage debe llamar al método `Security.allowDomain()` para permitir el dominio del entorno limitado externo. Para obtener más información, consulte "Controles de autor (desarrollador)" en la página 725.

La propiedad `frameRate` es especial, ya que cualquier archivo SWF puede leerla. Sin embargo, sólo pueden cambiar la propiedad los archivos que se encuentran en el entorno limitado de seguridad del propietario de Stage (o los que han obtenido permiso a través de una llamada al método `Security.allowDomain()`).

También hay restricciones relativas a los métodos `removeChildAt()` y `swapChildrenAt()` del objeto Stage, pero son distintas a las demás restricciones. En lugar de tener que estar en el mismo dominio que el propietario de Stage, para llamar a estos métodos, el código debe estar en el mismo dominio que el propietario de los objetos secundarios afectados, o bien los objetos secundarios pueden llamar al método `Security.allowDomain()`.

Recorrido de la lista de visualización

Existen restricciones con respecto a la capacidad de un archivo SWF para acceder a los objetos de visualización cargados desde otros entornos limitados. Para que un archivo SWF pueda acceder a un objeto de visualización creado por otro archivo SWF en un entorno limitado distinto, el archivo SWF al que se accede debe llamar al método `Security.allowDomain()` para permitir el acceso desde el dominio del archivo SWF que accede. Para obtener más información, consulte "Controles de autor (desarrollador)" en la página 725.

Para acceder a un objeto Bitmap cargado por un objeto Loader, debe existir un archivo de política URL en el servidor de origen del archivo de imagen y debe conceder permiso al dominio del archivo SWF que intenta acceder al objeto Bitmap (consulte "Controles de sitio Web (archivos de política)" en la página 721).

El objeto LoaderInfo correspondiente a un archivo cargado (y al objeto Loader) contiene las tres propiedades siguientes, que definen la relación entre el objeto cargado y el objeto Loader: `childAllowsParent`, `parentAllowsChild` y `sameDomain`.

Seguridad de eventos

Los eventos relacionados con la lista de visualización tienen limitaciones de acceso de seguridad que dependen del entorno limitado donde se encuentre el objeto de visualización que distribuye el evento. Un evento de la lista de visualización tiene fases de propagación y captura (descritas en "Gestión de eventos" en la página 254). Durante las fases de propagación y captura, un evento migra del objeto de visualización de origen a través de los objetos de visualización principales en la lista de visualización. Si un objeto principal se encuentra en un entorno limitado de seguridad distinto al del objeto de visualización de origen, la fase de captura y propagación se detiene debajo de dicho objeto principal, a menos que exista una confianza mutua entre el propietario del objeto principal y el propietario del objeto de origen. Esta confianza mutua se obtiene del siguiente modo:

1. El archivo SWF propietario del objeto principal debe llamar al método `Security.allowDomain()` para confiar en el dominio del archivo SWF propietario del objeto de origen.
2. El archivo SWF propietario del objeto de origen debe llamar al método `Security.allowDomain()` para confiar en el dominio del archivo SWF propietario del objeto principal.

El objeto LoaderInfo correspondiente a un archivo cargado (y al objeto Loader) contiene las dos propiedades siguientes, que definen la relación entre el objeto cargado y el objeto Loader: `childAllowsParent` y `parentAllowsChild`.

En los eventos distribuidos desde objetos que no sean objetos de visualización, no se realizan comprobaciones de seguridad.

Acceso a medios cargados como datos

Para acceder a los datos cargados, se utilizan métodos como `BitmapData.draw()` y `SoundMixer.computeSpectrum()`. De forma predeterminada, un archivo SWF de un entorno limitado de seguridad no puede obtener datos de píxeles ni datos de audio de objetos gráficos o de audio representados o reproducidos por medios cargados en otro entorno limitado. Sin embargo, se pueden utilizar los siguientes métodos para conceder este permiso:

- En un archivo SWF cargado, llame al método `Security.allowDomain()` para permitir que los datos puedan acceder a los archivos SWF de otros dominios.

- En una imagen, sonido o vídeo cargado, añada un archivo de política URL en el servidor del archivo cargado. Este archivo de política debe conceder acceso al dominio del archivo SWF que intenta llamar a los métodos `BitmapData.draw()` o `SoundMixer.computeSpectrum()` para extraer datos del archivo.

En las siguientes secciones se proporcionan detalles sobre el acceso a datos de mapa de bits, sonido y vídeo.

Acceso a datos de mapa de bits

El método `draw()` de un objeto BitmapData permite dibujar en el objeto BitmapData los píxeles de cualquier objeto de visualización que se están mostrando. Podrían ser los píxeles de un objeto MovieClip, un objeto Bitmap o cualquier objeto de visualización. Para que el método `draw()` dibuje píxeles en el objeto BitmapData, deben cumplirse las siguientes condiciones:

- En el caso de un objeto de origen que no sea un mapa de bits cargado, el objeto de origen y (en el caso de un objeto Sprite o MovieClip) todos sus objetos secundarios deben proceder del mismo dominio que el objeto que realiza la llamada al método `draw()`, o bien deben incluirse en un archivo SWF que sea accesible para el llamador mediante una llamada al método `Security.allowDomain()`.
- En el caso de un objeto de origen de mapa de bits cargado, el objeto de origen debe proceder del mismo dominio que el objeto que realiza la llamada al método `draw()` o su servidor de origen debe incluir un archivo de política URL que conceda permiso al dominio que realiza la llamada.

Si no se cumplen estas condiciones, se emite una excepción SecurityError.

Cuando se carga una imagen mediante el método `load()` de la clase Loader, se puede especificar un parámetro `context`, que es un objeto LoaderContext. Si se establece la propiedad `checkPolicyFile` del objeto LoaderContext en `true`, Flash Player comprueba si hay un archivo de política entre dominios en el servidor desde el cual se carga la imagen. Si hay un archivo de política y éste admite el dominio del archivo SWF que realiza la carga, el archivo podrá acceder a los datos del objeto Bitmap; en caso contrario, se denegará el acceso.

También se puede especificar una propiedad `checkPolicyFile` en una imagen cargada a través de una etiqueta `<img>` en un campo de texto. Para obtener información más detallada, consulte "Carga de archivos SWF e imágenes mediante la etiqueta <img> de un campo de texto" en la página 731.

Acceso a datos de sonido

Las siguientes API de ActionScript 3.0 relacionadas con el sonido tienen restricciones de seguridad:

- El método `SoundMixer.computeSpectrum()`: siempre se permite en archivos SWF que se encuentran en el mismo entorno limitado de seguridad que el archivo de sonido. Para los archivos de otros entornos limitados existen comprobaciones de seguridad.
- El método `SoundMixer.stopAll()`: siempre se permite en archivos SWF que se encuentran en el mismo entorno limitado de seguridad que el archivo de sonido. Para los archivos de otros entornos limitados existen comprobaciones de seguridad.
- La propiedad `id3` de la clase Sound: siempre se permite en archivos SWF que se encuentran en el mismo entorno limitado de seguridad que el archivo de sonido. Para los archivos de otros entornos limitados existen comprobaciones de seguridad.

Cada sonido tiene dos tipos de entornos limitados asociados, que son un entorno limitado de contenido y un entorno limitado de propietario:

- El dominio de origen del sonido determina el entorno limitado de contenido que, a su vez, determina si los datos pueden extraerse del sonido a través de la propiedad `id3` del sonido y a través del método `SoundMixer.computeSpectrum()`.

- El objeto que inició el sonido que se reproduce determina el entorno limitado de propietario que, a su vez, determina si el sonido puede detenerse a través del método `SoundMixer.stopAll()`.

Cuando se carga el sonido mediante el método `load()` de la clase Sound, se puede especificar un parámetro `context`, que es un objeto SoundLoaderContext. Si establece la propiedad `checkPolicyFile` del objeto SoundLoaderContext como `true`, Flash Player comprobará la existencia de un archivo de política URL en el servidor desde el que se carga el sonido. Si hay un archivo de política y éste admite el dominio del archivo SWF que realiza la carga, el archivo podrá acceder a la propiedad `id` del objeto Sound; en caso contrario, no será posible. Además, si se establece un valor de la propiedad `checkPolicyFile`, se puede activar el método `SoundMixer.computeSpectrum()` para sonidos cargados.

Se puede utilizar el método `SoundMixer.areSoundsInaccessible()` para saber si una llamada al método `SoundMixer.stopAll()` no va a detener todos los sonidos porque el que realiza la llamada no puede acceder al entorno limitado de uno o varios propietarios de sonidos.

Llamar al método `SoundMixer.stopAll()` permite detener estos sonidos cuyo entorno limitado de propietario es el mismo que el que realiza la llamada a `stopAll()`. También detiene los sonidos que han empezado a reproducirse porque unos archivos SWF realizaron una llamada al método `Security.allowDomain()` para permitir el acceso por parte del dominio del archivo SWF que realiza la llamada al método `stopAll()`. Los demás sonidos no se detienen y su presencia puede revelarse mediante una llamada al método `SoundMixer.areSoundsInaccessible()`.

Para llamar al método `computeSpectrum()`, es necesario que cada sonido que se esté reproduciendo se encuentre en el mismo entorno limitado que el objeto que realiza la llamada al método o que proceda de un origen que haya concedido permiso al entorno limitado del que realiza la llamada; en caso contrario, se emite una excepción SecurityError. En el caso de los sonidos cargados desde sonidos incorporados en una biblioteca de un archivo SWF, el permiso se concede a través de una llamada al método `Security.allowDomain()` en el archivo SWF cargado. En el caso de los sonidos que no proceden de archivos SWF (procedentes de archivos MP3 cargados o archivos de vídeo) un archivo de política URL en el servidor de origen concede acceso a los datos de los medios cargados. No se puede utilizar el método `computeSpectrum()` si un sonido se carga desde flujos RTMP.

Para obtener más información, consulte "Controles de autor (desarrollador)" en la página 725 y "Controles de sitio Web (archivos de política)" en la página 721

Acceso a datos de vídeo

Se puede utilizar el método `BitmapData.draw()` para capturar los datos de píxeles del fotograma actual de un vídeo.

Hay dos tipos distintos de vídeo:

- Vídeo RTMP
- Vídeo progresivo, que se carga desde un archivo FLV sin un servidor RTMP

No se puede utilizar el método `BitmapData.draw()` para acceder al vídeo RTMP.

Cuando se llama al método `BitmapData.draw()` con vídeo progresivo como valor del parámetro `source`, el que realiza la llamada a `BitmapData.draw()` debe encontrarse en el mismo entorno limitado que el archivo FLV, o el servidor del archivo FLV debe tener un archivo de política que conceda permiso al dominio del archivo SWF que realiza la llamada. Para solicitar la descarga del archivo de política, debe establecerse la propiedad `checkPolicyFile` del objeto NetStream en `true`.

Carga de datos

Los archivos SWF pueden cargar datos de servidores en ActionScript y enviar datos de ActionScript a servidores. La carga de datos es una operación distinta de la carga de medios, ya que la información cargada aparece directamente en ActionScript en lugar de mostrarse como medios. Por lo general, los archivos SWF pueden cargar datos de sus propios dominios. No obstante, suelen requerir archivos de política para cargar datos desde otros dominios (consulte "Controles de sitio Web (archivos de política)" en la página 721).

Utilización de URLLoader y URLStream

Se pueden cargar datos como, por ejemplo, un archivo XML o un archivo de texto. Los métodos `load()` de las clases URLLoader y URLStream se rigen por los permisos del archivo de política URL.

Si se utiliza el método `load()` para cargar contenido de un dominio ajeno al del archivo SWF que realiza la llamada al método, Flash Player comprueba si hay un archivo de política URL en el servidor de los activos cargados. Si hay un archivo de política y concede acceso al dominio del archivo SWF que realiza la carga, se pueden cargar los datos.

Conexión a sockets

De forma predeterminada, Flash Player busca un archivo de política de socket disponible en el puerto 843. Tal y como sucede con los archivos de política URL, este archivo se denomina*archivo maestro de política*.

Cuando se introdujeron por primera vez los archivos de política en Flash Player 6, no se admitían los archivos de política de socket. Las conexiones a servidores de socket se autorizaban a través de un archivo de política desde la ubicación predeterminada en un servidor HTTP del puerto 80 del mismo host que el servidor de socket. Flash Player 9 todavía admite esta capacidad, pero Flash Player 10 no. En Flash Player 10, sólo los archivos de política de socket pueden autorizar las conexiones de socket.

Al igual que los archivos de política URL, los archivos de política de socket admiten una sentencia de metapolítica que especifica los puertos que pueden servir los archivos de política. Sin embargo, en lugar de "master-only", la metapolítica predeterminada de los archivos de política de socket es "all". Es decir, a no ser que el archivo maestro de política especifique una configuración más restrictiva, Flash Player asumirá que cualquier socket del host puede servir un archivo de política de socket.

De forma predeterminada, el acceso a conexiones de socket y socket XML está desactivado, incluso si el socket al que se conecta se encuentra en el dominio al que pertenece el archivo SWF. El acceso a nivel de socket es posible si se sirve un archivo de política de socket desde cualquiera de las ubicaciones siguientes:

- Puerto 843 (la ubicación del archivo maestro de política)
- El mismo puerto que la conexión de socket principal
- un puerto diferente a la conexión de socket principal

De forma predeterminada, Flash Player busca un archivo de política de socket en el puerto 843 y en el mismo puerto que utiliza la conexión de socket principal. Para servir un archivo de política de socket desde un puerto diferente, el archivo SWF debe llamar a `Security.loadPolicyFile()`.

Los archivos de política de socket presentan la misma sintaxis que los archivos de política URL, salvo por el hecho de que también deben especificar los puertos a los que concede acceso. Cuando un archivo de política de socket procede de un número de puerto inferior a 1024, dicho archivo puede conceder acceso a cualquier puerto; cuando un archivo de política procede del puerto 1024 o superior, sólo puede conceder acceso a otros puertos 1024 y superiores. Los puertos permitidos se especifican en el atributo `to-ports` de la etiqueta `<allow-access-from>`. Se aceptan como valor los números de puerto únicos, los intervalos de puertos y los comodines.

A continuación se muestra un ejemplo de un archivo de política de socket:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- Policy file for xmlsocket://socks.mysite.com -->
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="507" />
    <allow-access-from domain="*.example.com" to-ports="507,516" />
    <allow-access-from domain="*.example.org" to-ports="516-523" />
    <allow-access-from domain="adobe.com" to-ports="507,516-523" />
    <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

Para recuperar un archivo de política de socket del puerto 843 o del mismo puerto que utiliza la conexión de socket principal, llame al método `Socket.connect()` o `XMLSocket.connect()`. Flash Player comprueba en primer lugar la presencia del archivo maestro de política en el puerto 843. Si encuentra alguno, comprobará si éste contiene una sentencia de metapolítica que prohíba los archivos de política de socket en el puerto de destino. Si el acceso no está prohibido, Flash Player buscará en primer lugar la sentencia `allow-access-from` en el archivo maestro de política. Si no encuentra ninguno, entonces buscará un archivo de política de socket en el mismo puerto que utiliza la conexión de socket principal.

Para recuperar un archivo de política de socket de una ubicación diferente, primero llame al método `Security.loadPolicyFile()` con la sintaxis especial `"xmlsocket"`, como en el ejemplo siguiente:

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Llame al método `Security.loadPolicyFile()` antes de llamar al método `Socket.connect()` o `XMLSocket.connect()`. Flash Player espera entonces hasta completar la solicitud del archivo de política antes de decidir si permite o no la conexión principal. Sin embargo, si el archivo maestro de política especifica que la ubicación de de destino no puede servir archivos de política, la llamada a `loadPolicyFile()` no será efectiva, incluso si hay un archivo de política en dicha ubicación.

Si se implementa un servidor de socket y se necesita proporcionar un archivo de política de socket, se debe decidir entre proporcionar el archivo de política a través del mismo puerto que acepta conexiones principales o utilizar otro puerto. En cualquier caso, para poder enviar una respuesta, el servidor deberá esperar a la primera transmisión de su cliente.

Cuando Flash Player solicita un archivo de política, siempre transmite la siguiente cadena en cuanto se establece una conexión:

```
<policy-file-request/>
```

Cuando el servidor recibe esta cadena, puede transmitir el archivo de política. La petición de Flash Player siempre termina con un byte null, y la respuesta del servidor debe terminar con otro byte del mismo tipo.

No cabe esperar que se pueda reutilizar la misma conexión para una solicitud de archivo de política y una conexión principal; cierre la conexión después de transmitir el archivo de política. De lo contrario, Flash Player cierra la conexión del archivo de política antes de volver a conectar para configurar la conexión principal.

Envío de datos

El envío de datos se produce cuando el código ActionScript de un archivo SWF envía datos a un servidor o recurso. El envío de datos siempre se permite en archivos SWF del dominio de red. Un archivo SWF local puede enviar datos a direcciones de la red únicamente si se encuentra en el entorno limitado local de confianza o en el entorno limitado local con acceso a la red. Para obtener más información, consulte “Entornos limitados locales” en la página 716.

Se puede utilizar la función `flash.net.sendToURL()` para enviar datos a un URL. Otros métodos también envían peticiones a URL. Algunos de estos métodos son los métodos de carga como `Loader.load()` y `Sound.load()`, y los métodos de carga de datos como `URLLoader.load()` y `URLStream.load()`.

Carga y descarga de archivos

El método `FileReference.upload()` inicia la carga de un archivo seleccionado por un usuario en un servidor remoto. Se debe llamar al método `FileReference.browse()` o `FileReferenceList.browse()` antes de llamar al método `FileReference.upload()`.

El código ActionScript que inicia el método `FileReference.browse()` o `FileReferenceList.browse()` sólo puede llamarse como respuesta a un evento de ratón o de teclado. Si se llama en otras situaciones, Flash Player 10 y versiones posteriores emitirán una excepción.

Al llamar al método `FileReference.download()`, se abre un cuadro de diálogo en el que el usuario puede descargar un archivo desde un servidor remoto.

Nota: *si el servidor requiere autenticación del usuario, sólo los archivos que se ejecutan en un navegador, es decir que utilizan el plug-in de navegador o controles ActiveX, pueden mostrar un cuadro de diálogo para pedir al usuario un nombre de usuario y una contraseña para la autenticación, y sólo para las descargas. Flash Player no permite realizar cargas en servidores que requieran autenticación de usuario.*

Las cargas y descargas no se permiten si el archivo SWF que realiza la llamada se encuentra en el entorno limitado local con sistema de archivos.

De forma predeterminada, un archivo SWF no puede realizar cargas ni descargas en un servidor ajeno. Un archivo SWF puede realizar cargas y descargas en otro servidor, si dicho servidor proporciona un archivo de política que conceda permiso al dominio del archivo SWF que realiza la llamada.

Carga de contenido incorporado de archivos SWF importados en un dominio de seguridad

Cuando se carga un archivo SWF, se puede establecer el parámetro `context` del método `load()` del objeto Loader que se utiliza para cargar el archivo. Este parámetro es un objeto LoaderContext. Si se establece la propiedad `securityDomain` de este objeto LoaderContext como `Security.currentDomain`, Flash Player comprueba si hay un archivo de política URL en el servidor del archivo SWF cargado. Si hay un archivo de política y concede acceso al dominio del archivo SWF que realiza la carga, se puede cargar el archivo SWF como medios importados. De este modo, el archivo que realiza carga puede obtener acceso a los objetos de la biblioteca del archivo SWF.

Otra forma de que un archivo SWF pueda acceder a las clases de los archivos SWF cargados de otros entornos limitados de seguridad es hacer que el archivo SWF cargado llame al método `Security.allowDomain()` para conceder acceso al dominio del archivo SWF que realiza la llamada. Se puede añadir la llamada al método `Security.allowDomain()` al método constructor de la clase principal del archivo SWF cargado y luego hacer que el archivo SWF que realiza la carga añada un detector de eventos para responder al evento `init` distribuido por la propiedad `contentLoaderInfo` del objeto Loader. Cuando se distribuye este evento, el archivo SWF cargado ha llamado al método `Security.allowDomain()` en el método constructor y las clases del archivo SWF cargado están disponibles para el archivo SWF que realiza la carga. El archivo SWF que realiza la carga puede recuperar las clases del archivo SWF cargado a través de una llamada a `Loader.contentLoaderInfo.applicationDomain.getDefinition()`.

Trabajo con contenido heredado

En Flash Player 6, el dominio utilizado para determinada configuración de Flash Player se basaba en el fragmento final del dominio del archivo SWF. Esta configuración incluye la configuración para permisos de cámaras y micrófonos, las cuotas de almacenamiento y el almacenamiento de objetos compartidos persistentes.

Si el dominio de un archivo SWF contiene más de dos segmentos, como en www.example.com, se quita el primer segmento del dominio (www) y se utiliza el fragmento restante. De este modo, en Flash Player 6, tanto www.example.com como store.example.com utilizan example.com como dominio para esta configuración. De forma análoga, tanto www.example.co.uk como store.example.co.uk utilizan example.co.uk como dominio para esta configuración. Esto puede originar problemas cuando archivos SWF de dominios no relacionados, como example1.co.uk y example2.co.uk, tienen acceso a los mismos objetos compartidos.

En Flash Player 7 y posterior, la configuración del reproductor se elige de forma predeterminada según el dominio exacto de un archivo SWF. Por ejemplo, un archivo SWF del dominio www.example.com utilizará la configuración del reproductor para www.example.com y un archivo SWF del dominio store.example.com utilizará la configuración del reproductor específica para store.example.com.

En un archivo SWF escrito en ActionScript 3.0, cuando `Security.exactSettings` se establece en `true` (valor predeterminado), Flash Player utiliza dominios exactos para la configuración del reproductor. Cuando se establece en `false`, Flash Player utiliza la configuración de dominio utilizada en Flash Player 6. Si se cambia el valor predeterminado de `exactSettings`, debe hacerse antes de que se produzca algún evento que requiera que Flash Player elija la configuración del reproductor, por ejemplo, al utilizar una cámara o micrófono, o al recuperar un objeto compartido persistente.

Si publica un archivo SWF de la versión 6 y crea objetos compartidos persistentes a partir de él, para recuperar estos objetos desde un archivo SWF que utilice ActionScript 3.0, debe establecer `Security.exactSettings` en `false` antes de llamar a `SharedObject.getLocal()`.

Configuración de permisos de LocalConnection

La clase LocalConnection permite desarrollar archivos SWF que pueden enviarse instrucciones entre sí. Los objetos LocalConnection sólo pueden comunicarse entre archivos SWF que se ejecuten en el mismo equipo cliente, aunque pueden estar ejecutándose en diferentes aplicaciones: por ejemplo, un archivo SWF que se esté ejecutando en un navegador y un archivo SWF que se esté ejecutando en un proyector.

En cada comunicación de LocalConnection hay un archivo SWF emisor y un archivo SWF detector. De forma predeterminada, Flash Player permite la comunicación de LocalConnection entre archivos SWF del mismo dominio. En archivos SWF de distintos entornos limitados, el detector debe dar permiso al emisor a través del método `LocalConnection.allowDomain()`. La cadena que pase como argumento al método `LocalConnection.allowDomain()` puede contener cualquiera de los elementos siguientes: nombres de dominio exactos, direcciones IP y el operador comodín `*`.

El método `allowDomain()` ha cambiado de la forma que tenía en ActionScript 1.0 y 2.0. En estas versiones anteriores `allowDomain()` era un método callback implementado por el usuario. En ActionScript 3.0, `allowDomain()` es un método incorporado de la clase LocalConnection que puede recibir llamadas. Con este cambio, el funcionamiento de `allowDomain()` es muy similar al de `Security.allowDomain()`.

Un archivo SWF puede utilizar la propiedad `domain` de la clase LocalConnection para determinar su dominio.

Control del acceso URL saliente

La creación de scripts y el acceso URL salientes (a través de URL HTTP, mailto:, etc.) se realizan a través de las siguientes API de ActionScript 3.0:

- La función `flash.system.fscommand()`
- El método `ExternalInterface.call()`
- La función `flash.net.navigateToURL()`

En los archivos SWF que se ejecutan localmente, las llamadas a estos métodos sólo se realizan correctamente si el archivo SWF y la página Web que lo contiene (si existe una) se encuentran en el entorno limitado de seguridad local de confianza. Las llamadas a estos métodos no se realizan correctamente si el contenido se encuentra en el entorno limitado local con acceso a la red o en el entorno limitado local con sistema de archivos.

En el caso de los archivos SWF que no se ejecutan de forma local, todas estas API se pueden comunicar con la página Web en la que están incorporadas, en función del valor del parámetro AllowScriptAccess que se describe a continuación. La función `flash.net.navigateToURL()` tiene la capacidad adicional de comunicarse con cualquier ventana o fotograma de navegador abierto, no sólo con la página en la que está incorporado el archivo SWF. Para obtener más información sobre esta funcionalidad, consulte "Utilización de la función navigateToURL()" en la página 742.

El parámetro `AllowScriptAccess` del código HTML que carga un archivo SWF controla la capacidad de realizar un acceso a URL saliente desde un archivo SWF. Defina este parámetro dentro de la etiqueta PARAM o EMBED. Si no se define ningún valor para `AllowScriptAccess`, el archivo SWF y la página HTML sólo se podrán comunicar si ambos se encuentran dentro del mismo dominio.

El parámetro `AllowScriptAccess` puede tener uno de los tres valores siguientes: `"always"`, `"sameDomain"` o `"never"`.

- Cuando `AllowScriptAccess` se establece como `"always"`, el archivo SWF se puede comunicar con la página HTML que lo incorpora, incluso cuando su dominio y el de la página son diferentes.
- Cuando `AllowScriptAccess` se establece como `"sameDomain"`, el archivo SWF se puede comunicar con la página HTML que lo incorpora sólo cuando su dominio y el de la página son iguales. Este es el valor predeterminado de `AllowScriptAccess`. Utilice esta configuración, o no defina un valor para `AllowScriptAccess`, para evitar que un archivo SWF alojado en un dominio acceda a un script de una página HTML perteneciente a otro dominio.
- Cuando `AllowScriptAccess` se establece como `"never"`, el archivo SWF no se puede comunicar con ninguna página HTML. El uso de este valor ha quedado desfasado en Adobe Flash CS4 Professional. No se recomienda y no debe ser necesario si no se proporcionan archivos SWF que no son de confianza desde su propio dominio. Si es preciso servir archivos SWF que no sean de confianza, Adobe recomienda crear un subdominio independiente y colocar en él todo ese contenido.

Este es un ejemplo de configuración de la etiqueta `AllowScriptAccess` en una página HTML para permitir el acceso URL saliente a un dominio diferente:

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.adobe.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,0,0'
height='100%' width='100%'>
<param name='AllowScriptAccess' value='always'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/getflashplayer'
src='MyMovie.swf' height='100%' width='100%' AllowScriptAccess='never'/>
</object>
```

Utilización de la función navigateToURL()

Además de la configuración de seguridad especificada por el parámetro `allowScriptAccess` descrito anteriormente, la función `navigateToURL()` tiene un segundo parámetro opcional: `target`. El parámetro `target` se puede utilizar para especificar el nombre de una ventana o fotograma HTML al que enviar la petición de URL. Las restricciones de seguridad adicional se aplican a dichas peticiones y varían en función de si `navigateToURL()` se utiliza como una sentencia de creación de scripts o como una sentencia de no creación de scripts.

En el caso de las sentencias de creación de scripts, como `navigateToURL("javascript: alert('Hello from Flash Player.')")`, es necesario tener en cuenta las reglas siguientes.

- Si el archivo SWF es un archivo de confianza local, la solicitud funcionará correctamente.
- Si el destino es la página HTML en la que se incorpora el archivo SWF, se aplicarán las reglas `allowScriptAccess` descritas anteriormente.
- Si el destino tiene contenido cargado del dominio al que pertenece el archivo SWF, la solicitud funcionará correctamente.
- Si el destino tiene contenido cargado de un dominio distinto al que pertenece el archivo SWF, y no se cumple ninguna de las condiciones anteriores, la solicitud no funcionará correctamente.

En el caso de las sentencias que crean scripts (como HTTP, HTTPS y `mailto:`), la solicitud no funcionará correctamente si se cumplen todas las condiciones siguientes:

- El destino es una de las palabras clave especiales `"_top"` o `"_parent"`, y
- el archivo SWF se encuentra en una página Web alojada en un dominio diferente, y
- el archivo SWF está incorporado con un valor para `allowScriptAccess` que no es `"always"`.

Para obtener más información

Para obtener más información sobre el acceso URL saliente, consulte las entradas siguientes en el *Referencia del lenguaje y componentes ActionScript 3.0*:

- La función `flash.system.fscommand()`
- El método `call()` de la clase ExternalInterface
- La función `flash.net.navigateToURL()`

Objetos compartidos

Flash Player proporciona la capacidad de utilizar *objetos compartidos*, que son objetos de ActionScript que persisten fuera de un archivo SWF, ya sea localmente en el sistema de archivos de un usuario o remotamente en un servidor RTMP. Los objetos compartidos, al igual que otros medios de Flash Player, se dividen en entornos limitados de seguridad. Sin embargo, el modelo de entorno limitado de los objetos compartidos es algo distinto, ya que los objetos compartidos no son recursos a los que se puede tener acceso más allá de los límites de los dominios. Por el contrario, los objetos compartidos siempre se recuperan de un almacén de objetos compartidos que es específico del dominio de cada archivo SWF que llama a los métodos de la clase SharedObject. Los almacenes de objetos compartidos suelen ser incluso más específicos que los dominios de los archivos SWF: de forma predeterminada, cada archivo SWF utiliza un almacén de objetos compartidos específico de su URL de origen completa.

Un archivo SWF puede utilizar el parámetro `localPath` de los métodos `SharedObject.getLocal()` y `SharedObject.getRemote()` para usar un almacén de objetos compartidos asociado solamente a una parte de su URL. De este modo, el archivo SWF puede permitir el uso compartido con otros archivos SWF de otros URL. Incluso si se pasa el valor `'/'` para el parámetro `localPath`, se sigue designando un almacén de objetos compartidos específico de su propio dominio.

Los usuarios pueden restringir el acceso a objetos compartidos utilizando el cuadro de diálogo Configuración de Flash Player o el Administrador de configuración. De forma predeterminada, pueden crearse objetos compartidos de un máximo de 100 KB de datos por dominio. Los usuarios con derechos administrativos y los usuarios pueden también limitar la capacidad de escribir en el sistema de archivos. Para obtener más información, consulte "Controles de administrador" en la página 718 y "Controles de usuario" en la página 720.

Para especificar que un objeto compartido sea seguro, se debe establecer el valor `true` para el parámetro `secure` del método `SharedObject.getLocal()` o del método `SharedObject.getRemote()`. Tenga en cuenta las siguientes cuestiones relativas al parámetro `secure`:

- Si el valor de este parámetro es `true`, Flash Player crea un nuevo objeto compartido seguro u obtiene una referencia a un objeto compartido seguro existente. Sólo pueden leer o escribir en este objeto compartido seguro archivos SWF enviados a través de HTTPS que llamen a `SharedObject.getLocal()` con el parámetro `secure` definido como `true`.
- Si el parámetro se establece en `false`, Flash Player crea un nuevo objeto compartido o bien obtiene una referencia a un objeto compartido existente que pueda leerse y escribirse con archivos SWF enviados mediante conexiones que no son HTTPS.

Si el archivo SWF que realiza la llamada no procede de un URL HTTPS, al especificar el valor `true` para el parámetro `secure` del método `SharedObject.getLocal()` o del método `SharedObject.getRemote()`, se emitirá una excepción SecurityError.

La elección de un almacén de objetos compartidos se basa en el URL de origen de un archivo SWF. Esto es cierto incluso en las dos situaciones en las que un archivo SWF no se origina en un simple URL: carga de importación y carga dinámica. La carga de importación hace referencia a la situación en la que se carga un archivo SWF con la propiedad `LoaderContext.securityDomain` establecida en `SecurityDomain.currentDomain`. En esta situación, el archivo SWF cargado tendrá un pseudoURL que empieza por el dominio del archivo SWF de carga, seguido del URL de origen real. La carga dinámica se refiere a la carga de un archivo SWF a través del método `Loader.loadBytes()`. En esta situación, el archivo SWF cargado tendrá un pseudoURL que empieza por el URL completo del archivo SWF que realiza la carga, seguido de un ID de entero. Tanto en la carga de importación como en la carga dinámica, el pseudoURL de un archivo SWF puede examinarse con la propiedad `LoaderInfo.url`. El pseudoURL se trata del mismo modo que un URL real, en lo que se refiere a la elección de un almacén de objetos compartidos. Se puede especificar un parámetro `localPath` de objeto compartido que utilice el pseudoURL parcialmente o en su totalidad.

Los usuarios y los administradores pueden optar por desactivar la utilización de *objetos compartidos de terceros*. Es la utilización de objetos compartidos por parte de cualquier archivo SWF que esté ejecutándose en un navegador Web, cuando el URL de origen de dicho archivo SWF procede de un dominio distinto al del URL que se muestra en la barra de direcciones del navegador. Los usuarios y administradores pueden optar por desactivar la utilización de objetos compartidos de terceros por motivos de privacidad, con el fin de evitar el seguimiento entre dominios. Para evitar esta restricción, quizá desee asegurarse de que cualquier archivo SWF que utilice objetos compartidos sólo se cargue en las estructuras de página HTML que garantizan que el archivo SWF procede del mismo dominio que se muestra en la barra de direcciones del navegador. Cuando se intentan utilizar objetos compartidos desde un archivo SWF de terceros y la utilización de objetos compartidos de terceros está desactivada, los métodos `SharedObject.getLocal()` y `SharedObject.getRemote()` devuelven `null`. Para obtener más información, consulte www.adobe.com/es/products/flashplayer/articles/thirdpartylso.

Acceso a la cámara, el micrófono, el portapapeles, el ratón y el teclado

Cuando un archivo SWF intenta acceder a la cámara o micrófono de un usuario a través de los métodos `Camera.get()` o `Microphone.get()`, Flash Player muestra un cuadro de diálogo Privacidad que permite al usuario autorizar o denegar el acceso a la cámara o al micrófono. El usuario y el usuario administrador también pueden desactivar el acceso a la cámara para cada sitio o de forma global, a través de controles en el archivo mms.cfg, la interfaz de usuario Configuración y el Administrador de configuración (consulte "Controles de administrador" en la página 718 y "Controles de usuario" en la página 720). Con restricciones de usuario, cada uno de los métodos `Camera.get()` y `Microphone.get()` devuelve un valor `null`. Se puede utilizar la propiedad `Capabilities.avHardwareDisable` para determinar si el administrador ha prohibido (`true`) o permitido (`false`) el acceso a la cámara y el micrófono.

El método `System.setClipboard()` permite que un archivo SWF sustituya el contenido del portapapeles por una cadena de texto normal. Esto no supone ningún riesgo de seguridad. Para protegerse del riesgo que supone cortar o copiar contraseñas y otros datos confidenciales en el portapapeles, no existe ningún método `getClipboard()`.

Una aplicación que se ejecute en Flash Player sólo puede controlar los eventos de teclado y ratón que sucedan en su ámbito. El contenido que se ejecuta en Flash Player no puede detectar eventos de teclado o ratón en otra aplicación.

Índice

E

F

N

O

P

R

S

T

U

Z